AN AUTOMATED GRADING AND FEEDBACK SYSTEM FOR A COMPUTER LITERACY COURSE

A Thesis
by
BAHAREH AKHTAR

Submitted to the Graduate School
at Appalachian State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

December 2015
Department of Computer Science

AN AUTOMATED GRADING AND FEEDBACK SYSTEM FOR A COMPUTER LITERACY COURSE

A Thesis
by
BAHAREH AKHTAR
December 2015

APPROVED BY:

James B. Fenwick Jr.
Chairperson, Thesis Committee

Rahman Tashakkori
Member, Thesis Committee

James T. Wilkes
Member, Thesis Committee

James T. Wilkes
Chairperson, Department of Computer Science

Max C. Poole, Ph.D.
Dean, Cratis D. Williams School of Graduate Studies

Abstract

An Automated Grading And Feedback System for a Computer Literacy Course

Bahareh Akhtar

B.S., Buali Sina University

M.S., Appalachian State University

Chairperson: Jay Fenwick

Computer Science departments typically offer a computer literacy course that targets a general lay audience. At Appalachian State University, this course is *CS1410 - Introduction to Computer Applications*. Many computer literacy courses have students work with various desktop and web-based software applications, including standard office applications. CS1410 strives to have students use well known applications in new and challenging ways, as well as exposing them to some unfamiliar applications. These courses can draw large enrollments which impacts efficient and consistent grading.

This thesis describes the development and successful deployment of the Automated Grading And Feedback (AGAF) system for CS1410. Specifically, a suite of automated grading tools targeting the different types of CS1410 assignments has been built. The AGAF system tools have been used on actual CS1410 submissions and the resulting grades were verified. AGAF tools exist for Microsoft Office assignments requiring students to upload a submission file. Another AGAF tool accepts a student "online text submission" where the text encodes the URL of a Survey Monkey survey and a blog. Other CS1410 assignments require students to upload an image file.

AGAF can process images in multiple ways, including decoding of a QR two-dimensional barcode and identification of an expected image pattern.

Acknowledgments

Firstly, I would like to thank my great advisor, Dr. Jay Fenwick, for his guidance and support of my research and studies. I am thankful for his patience to assist my research and writing of this thesis. Without his motivation and encouragement during these years I may have given up long ago. It was through his understanding and kindness that I was encouraged to keep up working and complete my graduate degree.

I would also like to thank my thesis committee, Dr. Rahman Tashakkori and Dr. James Wilkes for their time and effort. They have both been great resources and their contributions are greatly appreciated.

My sincere thanks also goes to Tashakkori's family specially Dr.Sharareh Nikbakht for always being there for me when my life was pulling me down. Their endless kindness and hospitality that each of them showed me in these several years is unforgettable. I doubt that I will ever be able to convey my appreciation fully, but I owe them my eternal gratitude.

Last but not least, I would like to thank my family: my parents and to my brothers and to my uncle for supporting me spiritually to my life in general.

Contents

Chapter 1 - Introduction

Computer Science departments typically offer a computer literacy course that targets a general lay audience. These courses can draw large enrollments which impacts efficient and consistent grading. There is a strong relationship between the number of students in a classroom and difficulties with grading and giving feedback to them [16]. Consistency in grading drops tremendously as the number of students enrolled in a class increases. Moreover, a problem-based learning pedagogy [25] introduces challenges in evaluating student work consistently and fairly [24].

Online classes are getting very popular in education [30]. They are effective and have high learner satisfaction [13]. This popularity can cause a large increase in enrollments of students in online classes, so providing consistent, timely, and detailed feedback to them is a serious challenge. Also, a tremendous amount of time can be spent in grading the high volume of student assignment submissions.

The Department of Computer Science at Appalachian State University offers a computer literacy course for non-majors. This course, *CS1410 - Introduction to Computer Applications*, typically serves between 80-120 students per semester and is delivered fully online. CS1410 uses a problem-based learning approach to help students learn about computer knowledge through the experience of solving problems.

CS1410 requires about 20 assignment submissions of various types including file uploads and online text entries. Grading is a human-centered, time-intensive process involving downloads of submission file(s), opening documents, and examination of document contents for various required elements.

There has always been a strong need to grade students with less human error and using more consistent methods. Pregent describes two grading strategies aimed at providing consistent, unbiased grading [20]. The "horizontal" grading technique grades one problem for all submissions; whereas, the "vertical" technique grades each submission for all problems. In the context of large, online enrollments, horizontal grading is problematic because it requires having dozens of open documents grading each element one at a time. Vertical grading is challenging in this context because of the time involved to get all student submissions graded in a single session. A primary challenge of either technique involves the timeliness of grading the high volume of student submissions. This grading challenge can lead to grading inconsistency or errors, notwithstanding the tedious and time-consuming nature of the work. A number of recent systems provide for automated feedback of computer program submissions. Some, like WEB-CAT, are standalone systems, and others, like Codeacademy work in conjunction with the programming environments providing instantaneous feedback. These systems inspired the idea for an automated grading and feedback system for CS1410. If a computer program file can be verified then why can't an Excel spreadsheet file?

This thesis presents an automated grading and feedback (AGAF) system for computer literacy courses, specifically for CS1410. AGAF is a suite of tools developed using different programming languages. The languages used are based on the context and requirements of the assignments. AGAF tools exist for Excel assignments requiring students to upload a submission file. Another AGAF tool accepts a student "text submission" where the text encodes the URL of a Survey Monkey survey. Other CS1410 assignments require students to upload an image file. AGAF can process images in multiple ways, including decoding of a QR two-dimensional barcode and the identification of an expected image pattern such as an activity completion "badge.

Chapter 2 provides background material on XML formats, and QR code decoder methods and extracted URL of QR codes that may benefit some readers. Chapter 3 presents related work particularly in the area of automated program analysis. Chapter 4 describes the details of the various AGAF tool design and implementations. Chapter 5 presents results of using AGAF to help grade CS1410 during the summer 2015 term. Chapter 6 presents concludes and identifies some areas of future work.

Chapter 2 - Background

The AGAF suite of grading tools employs a variety of technologies. Chapter 4 describes the implementation of the AGAF tools and assumes a working knowledge of some of these technologies. This chapter serves to provide this background knowledge; specifically, the internal formats of HTML web pages and how HTML web pages are loaded are described.

Modern web technology is a mixture of cooperative, interacting technologies. Three of the most fundamental ideas are: HTML, HTTP, and URLs [2]. Hyper Text Markup Language (HTML) is a language that was designed to represent documents for the World Wide Web [14]. Each HTML document consists of markup tags that are in charge of the appearance and content of pages displayed by browsers. Figure 2.1 on the left side shows the contents of a simple HTML file, and on the right side shows the browser rendering of this same file.

The nested block structure of an HTML document is visualized in Figure 2.1. Each block is delineated with starting and ending "tags." The nesting of tags is clearer in Figure 2.2, which uses pink boxes. The entire document is enclosed in the starting <HTML> and ending </HTML> tags. The document header is between the <HEAD> and </HEAD> tags, and then the body of document is between the <BODY> and </BODY> tags. The body of the provided example has a level-1 heading and a paragraph.

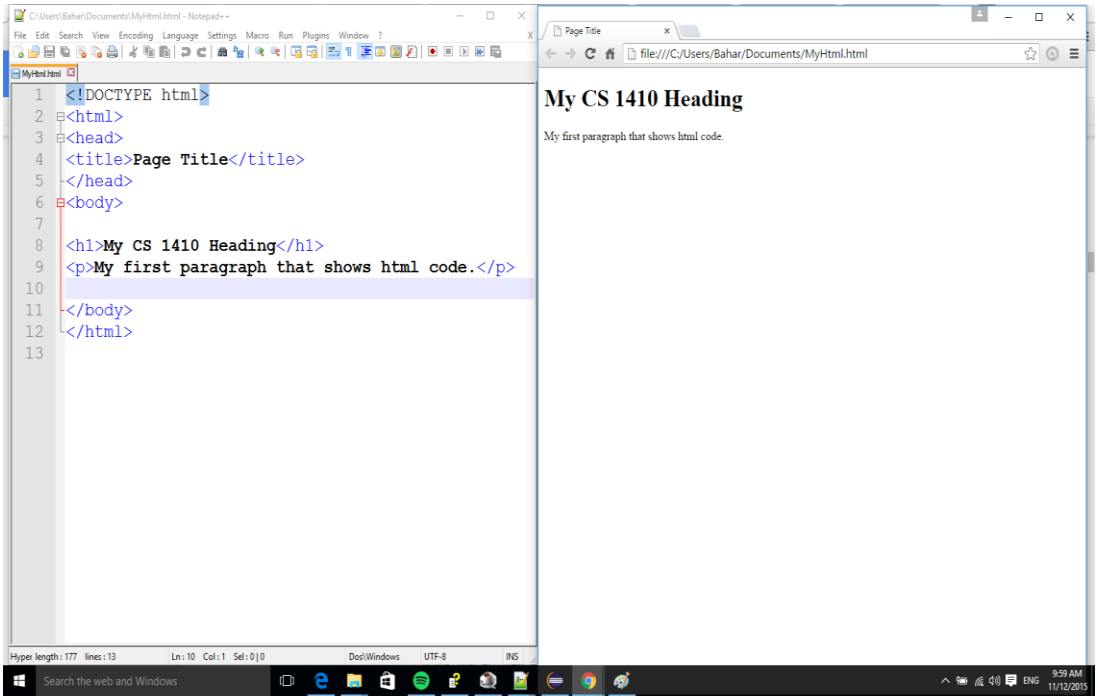


Figure 2.1: HTML file contents and Browser rendering of this file

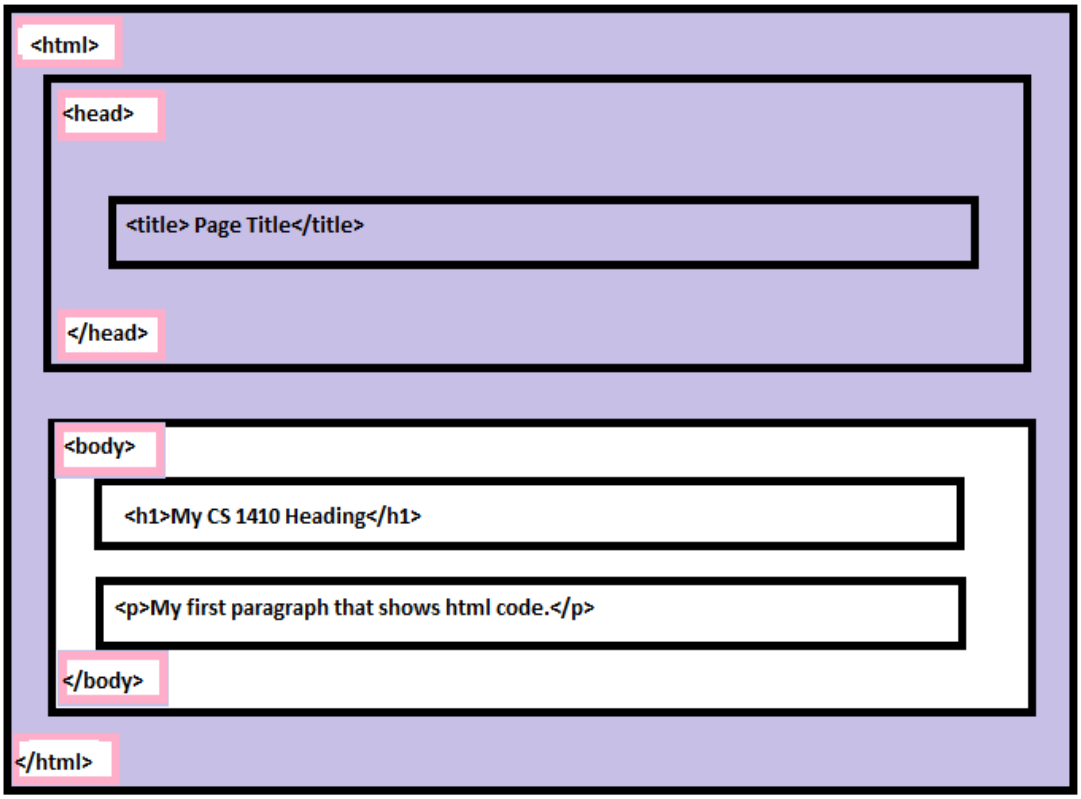


Figure 2.2: HTML page nested structure

HTML files are computer text files that are rendered, or displayed, by web browsers. Each HTML webpage has its own unique address that is requested by typing its address into a browser. After the request for a web page, the web server finds the web page file and sends it back to the user. The flow diagram of these transitions is shown in figure 2.3. The user initiates a request for a webpage from a browser application. The request is then routed through the Internet to the indicated web server that locates the requested page and returns it through the Internet to the requestor.

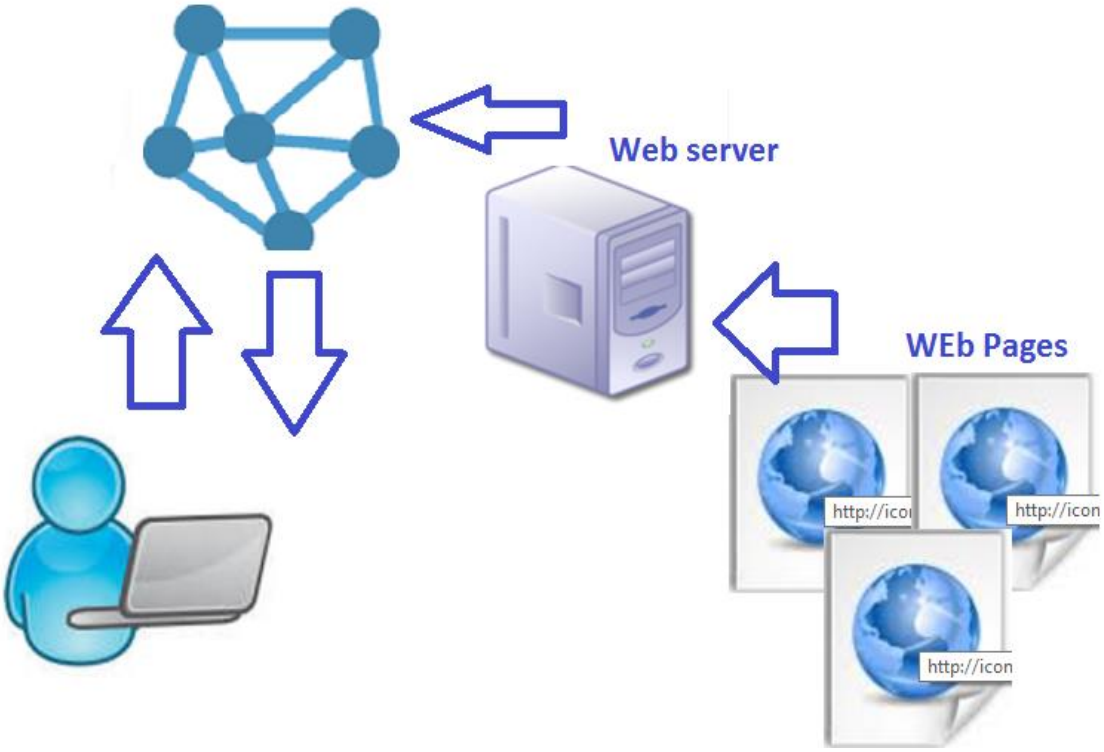


Figure 2.3: Flow diagram of web server and web pages

This communication between the requestor and the server is performed using the Hyper-Text Transfer Protocol (HTTP) [8]. HTTP clients send requests to hosts (servers) and get responses from the hosts. These requests are encoded as Uniform Resource

Locators (URLs). Figure 2.4 shows the structure of a URL in detail. Each URL contains the protocol, the host machine of the web server, and the path to the HTML resource file. The client (usually a browser) forms the URL and makes a protocol (HTTP) request to the web server at the host machine indicated. The web server then uses resource path portion of the URL to locate the desired resource (webpage). More dynamic web content is possible when the client can pass additional information to the web server via query parameters. When the desired resource is located, the web server returns the HTML webpage as an HTTP response. The response returns to the requestor and is handled appropriately, usually being rendered in a browser [6].

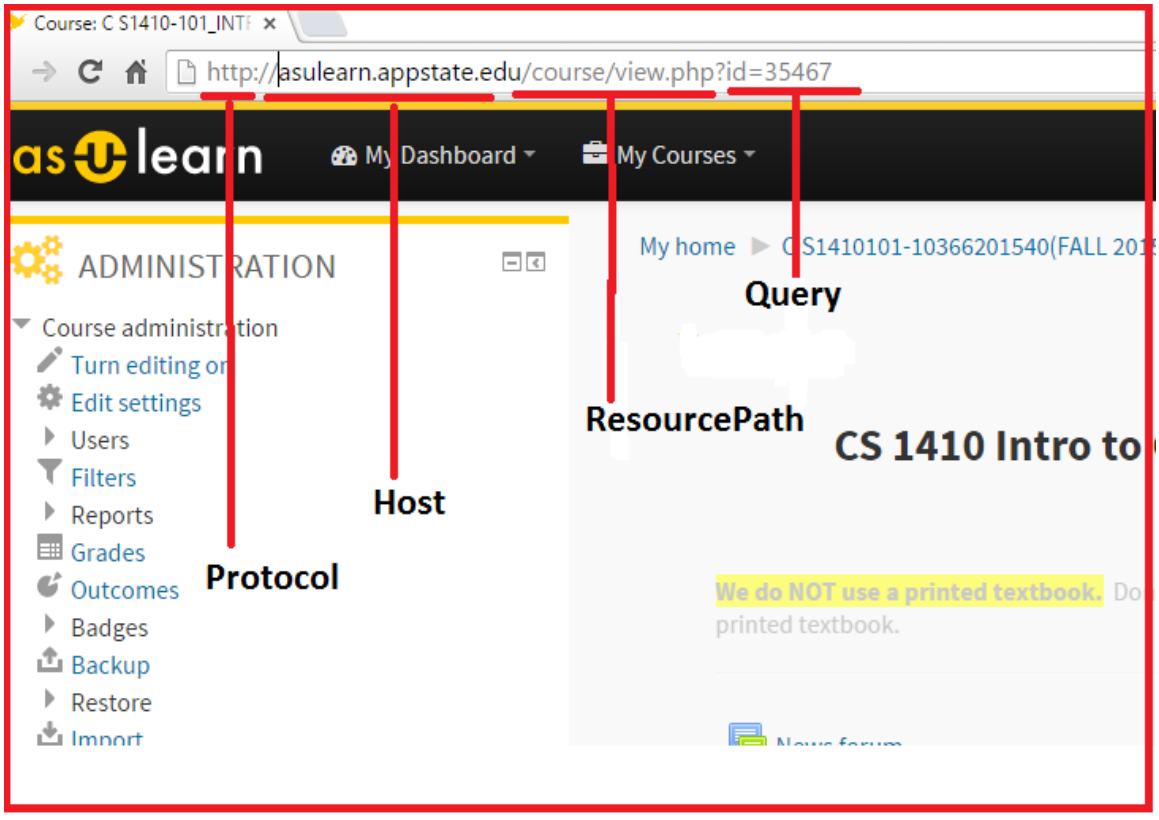


Figure 2.4: URL structure

The requestor of a webpage is usually a browser application. However, other programs can act as a client and make a request for a web page. “Page scraping” is a term used to describe when a non-browser application requests an HTML page from a web server in order to extract data from the web pages [21]. By knowing the HTML structure, a page scraper can look for desired elements. For example, we could create a page scraper to look for <H1> headings that have “CS 1410” in them, as shown in Figure 2.5.

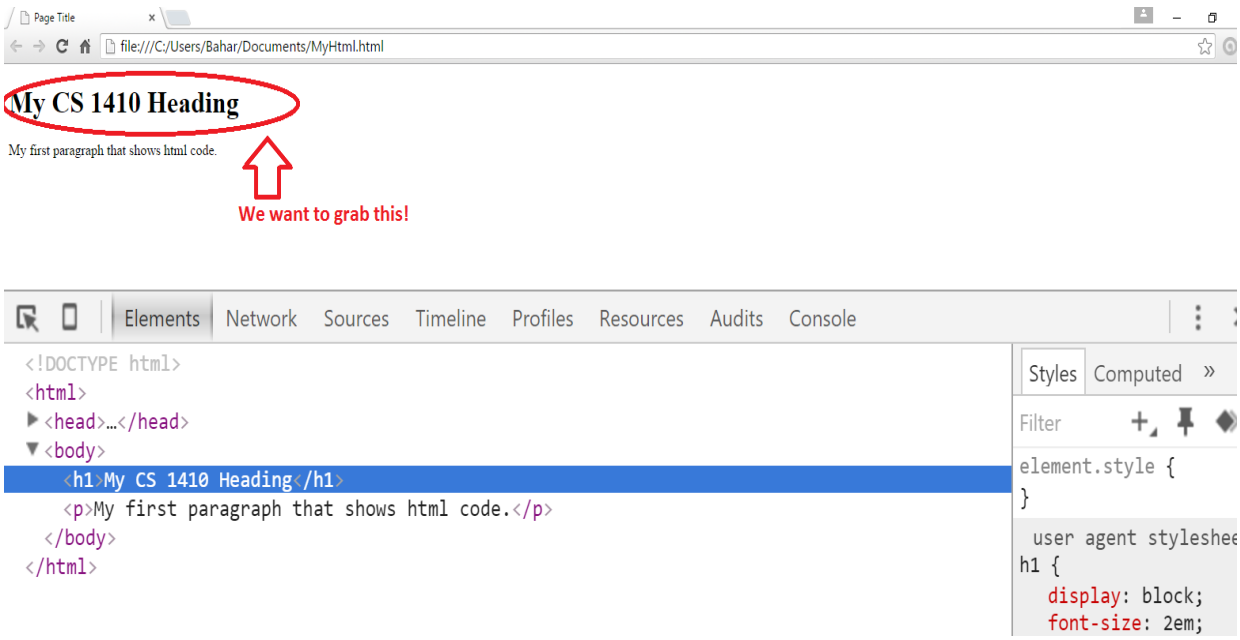


Figure 2.5: HTML structure

Page scrapers can be brittle applications breaking easily when the HTML changes even just a little. Sites that change their HTML pages frequently are considered volatile and can wreak havoc on a page scraper. For example, if the earlier <H1> heading were changed to an <H2> heading, a page scraper may not find its target

Chapter 3 - Related Work

This chapter provides descriptions of related efforts by other researchers. Section 3.1 has a focus on automated grading of programming assignments. Many computer literacy courses do not have students write programs in a traditional programming language, such as Java or Python, but the efforts in trying to build automated grading systems covers some similar ground. Moreover, it was these systems that inspired the idea for AGAF. Section 3.2 discusses a range of other techniques and tools.

3.1 – Automated grading of programming assignments

Automated systems targeting the grading of computer programs have a relatively long history. An early system was ASSYST, an automated system for assessment of programming [12]. ASSYST is not only an assessment tool but also helps with housekeeping tasks such as submission. This system benefits students by checking the efficiency and correctness of their assignments. The authors claim it has proven to be a very useful tool and helps improve the consistency of grading.

Web-CAT, the Web-based Center for Automated Testing, is a widely used open-source automated grading system for programming assignments and was developed by Virginia Tech researchers [27]. Students submit their programs via a web-based form to Web-CAT. The Web-CAT server then runs the student code against a set of test cases reporting the results of those tests back to the student. This approach scales well and

provides consistency through its test cases. Instructors can choose to have grades computed based on tests passing and failing or to just use the information for student feedback. This type of system has proven useful but crucially depends on well-crafted test cases.

A recent trend combining technology advancements and education is the development of massive open online courses (MOOCs) [5]. A couple of reasons for the rising popularity of MOOCs are their flexibility and immediate feedback. For MOOC programming courses, students enter code into a browser window and that code is automatically evaluated with results returned immediately to the student. Typically the student-to-teacher ratio in most MOOCs is very high, thus automated grading and personalized feedback are essential. Giving this type of feedback to each student is a challenge described by Nguyen [18], although recent advances in MOOC delivery are making some progress.

Kurnia et al. developed an Online Judge system that tests student programs submitted either in email or from a webpage form [16]. This system is a forebear of the Web-CAT system discussed previously. The Online Judge system builds the program and runs it against predefined test cases. The results of this testing are termed the "judgment," hence the name for the system.

3.2 Other related efforts

Perhaps most relevant to AGAF is the work on the automated assessment of Office applications by Zlatko and Green [31]. Their application as described only works for Excel assignments, which is just one AGAF component. Their approach utilizes the online quiz feature of a learning management system (i.e., Moodle) to provide the

automated grading and feedback. They distribute an Excel spreadsheet to students that has "hidden" worksheets, formulas, and macro functions. To answer questions in the automated quiz, students must try various techniques in the Excel workbook and copy/paste those functions/formulas they think are working properly. Some cells have values computed from the hidden material that students report in the quizzes also. The AGAF approach differs by having students create Excel workbooks in the usual way and submit that work for analysis.

Related to assignment grading of programming submissions is an analysis of assignments for plagiarism. Online courses are especially vulnerable to plagiarism since there is no witnessed assurance of student effort and there may be a large number of student submissions. Perhaps the best-known plagiarism detection system targeting program submissions is Aiken's Measure of Software Similarity (MOSS) [1]. The internal grammatical structure of the submission is compared against other submissions for similarity. The tool only reports a measure of similarity leaving it to the discretion of the user to make a determination of the reported similarity value.

Warne's thesis *"Experiences in Implementing Automated Evaluation and Grading in a MOOC Using a Behavior-driven Testing Framework"* explores a novel way of doing program grading [26]. A unique aspect of his approach is using the test-driven development philosophy as a behavioral testing framework. His system uses the RSpec (Request Specs) and Capybara features of the Ruby programming language to manage the test case development and deployment in its own mocking framework to test course assignments.

Williamson , Bennett, and Lazer proposed an Automated Scoring for the Assessment of Common Core Standards that uses artificial intelligence and computing technology [29]. Natural language processing (NLP) and image processing methods have become more accessible and therefore viable in systems such as these. By applying these new types of methods, analyzing the text of student assignments can be very precise. Currently, AGAF tools did not have the requirement to grade student text, for example the quality of a student blog post, but this related work presents an option should computer literacy courses wish to head in that direction.

Han's and Liu's work *"The Application of Natural Language Processing and Automated Scoring in Second Language Assessment"* also uses an NLP approach [10]. This application was developed to assess writing and speaking English tests. Using NLP techniques, the number of words in essays is counted and the length of silences and spoken words is measured automatically. This extracted dataset is measured for each type of test. Tree structures based on the semantic words in each assignment are computed and this is computed into a "word vector." This word vector is used to determine a level of correctness.

Adaptive learning is another field of study in artificial intelligence that uses computers to learn the correct pattern of answers. This method is applied in automated grading tools too. Kevin, Thomas, and Laurie described such an approach in their paper titled "Implementation of an Automated Grading System with an Adaptive Learning Component to Affect Student Feedback and Response Time" [15]. Adaptive learning is an unsupervised learning method that clusters the wrong answers and the correct answers. Their technique runs faster than other methods of artificial intelligence.

These newer technology approaches are not without critics however. There are still some errors because of the complexity of language processing. Perelman published a critique of an automated essay scoring paper [19]. He argues that human graders and software graders must be compared directly to validate research in this field.

Chapter 4 - The AGAF Tool Suite

The online computer literacy course at Appalachian State University, CS1410, requires about 20 assignment submissions of various types including file uploads and online text entries. To flexibly support this variety, the AGAF system is comprised of a suite of tools. AGAF tools exist Excel assignments that require students to upload a submission file. Another AGAF tool accepts a student "text submission" where the text encodes the URL of a Survey Monkey survey. Other CS1410 assignments require students to upload an image file. AGAF can process images in multiple ways, including decoding of a QR two-dimensional barcode and the identification of an expected image pattern such as an activity completion "badge. The AGAF tools are developed using different programming languages that are chosen based on the context and requirements of the assignments.

The first two sections below describe the overall system architecture and how an instructor "grades" an assignment using an AGAF tool. The remaining sections describe the AGAF tools implementations.

4.1 AGAF Tool Architecture

Different course assignments have students submit different types of items. Sometimes they submit "online text" using a box in the assignment submission page of the course management system. Other times they upload a file via the assignment

submission page. The CS1410 course at Appalachian State University uses a branded Moodle-based course management system (CMS) locally called ASULEARN. This CMS provides a method for the instructor to download into a folder on their computer all the student submissions. Each student submission is downloaded into a file of the same type; for example, text submissions are saved into text files and file submissions are saved as they are uploaded (e.g., PowerPoint files are saved as PPT or PPTX files). Each student submission file is named to contain information about the student, and the assignment. For example, student Jane Doe with CMS identifier 12345 submitting online text for an assignment might be saved into a text-based HTML file named:

```
Jane Doe_12345_assignsubmission_onlinetext.html
```

Similarly, Jane Doe's submission of a PowerPoint file she named "Computers.pptx" on her computer would be downloaded by the instructor into a file on their computer named:

```
Jane Doe_12345_assignsubmission_file_Computers.pptx.
```

All student submissions for each assignment are downloaded by the instructor into the same folder and the folder name encodes course and assignment information.

Figure 4.1 shows the overall architecture of the AGAF tool suite. The various grader application tools draw upon two inputs. One input is the folder containing all of the student submission files for an assignment. The second input is a comma-separated values (CSV) file that lists student names and CMS identifiers. Most CMS tools provide a mechanism for exporting student participant information into this format. The grader application tools use these inputs to perform their specific grading functions. They output

results that include numeric grades and textual feedback into another CSV file. Most CMS tools provide a mechanism to import grades via files of this format.

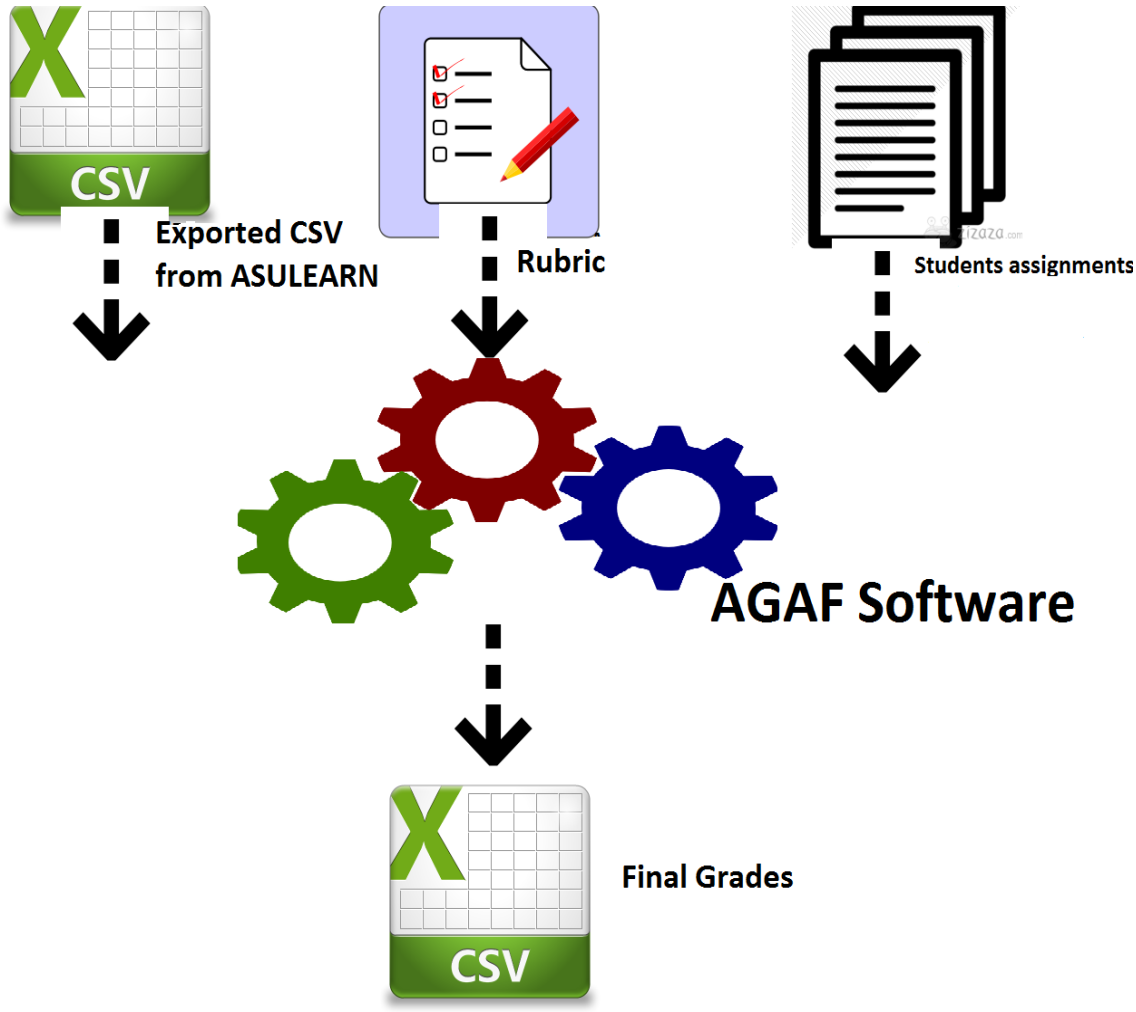


Figure 4.1: AGAF system architecture

4.2 Web-page Processing

A number of CS1410 assignments have students create work using a web-based tool. They then provide a URL link to their completed work. Specifically, the student uploads the URL as text into the course management system as “online text.” AGAF must read this text submission, interpreting it as a URL. Then AGAF must “view” the

student's work checking for various required elements. As a simple example, a student might be assigned to build a web page containing an image. A human grader would copy/paste the URL submission into a browser, wait for the page to load, then confirm that an image was rendered. As described in Chapter 2, a program examining the HTML of the web page is called "page scraping."

CS1410 has several assignments that require instructors to view a web site. The subsections below detail two such assignments. The first requires students to encode a URL into a QR two-dimensional bar code image. From this URL, the AGAF tool must ensure the required elements. The second CS1410 assignment has the user use a free version of the popular survey tool known as Survey Monkey. Students create a survey and provide a URL to their survey as an online text submission.

4.2.1 QR code digital business card

This assignment requires students to use the qrstuff.com web-based tools to create a "digital business card" that contains typical business card attributes including a picture. In addition, students must embed the URL of their digital business card into a QR code. They submit the resulting QR code image. Figure 4.2 shows an image from the assignment description that depicts a QR code and the digital business card that results from reading that code.

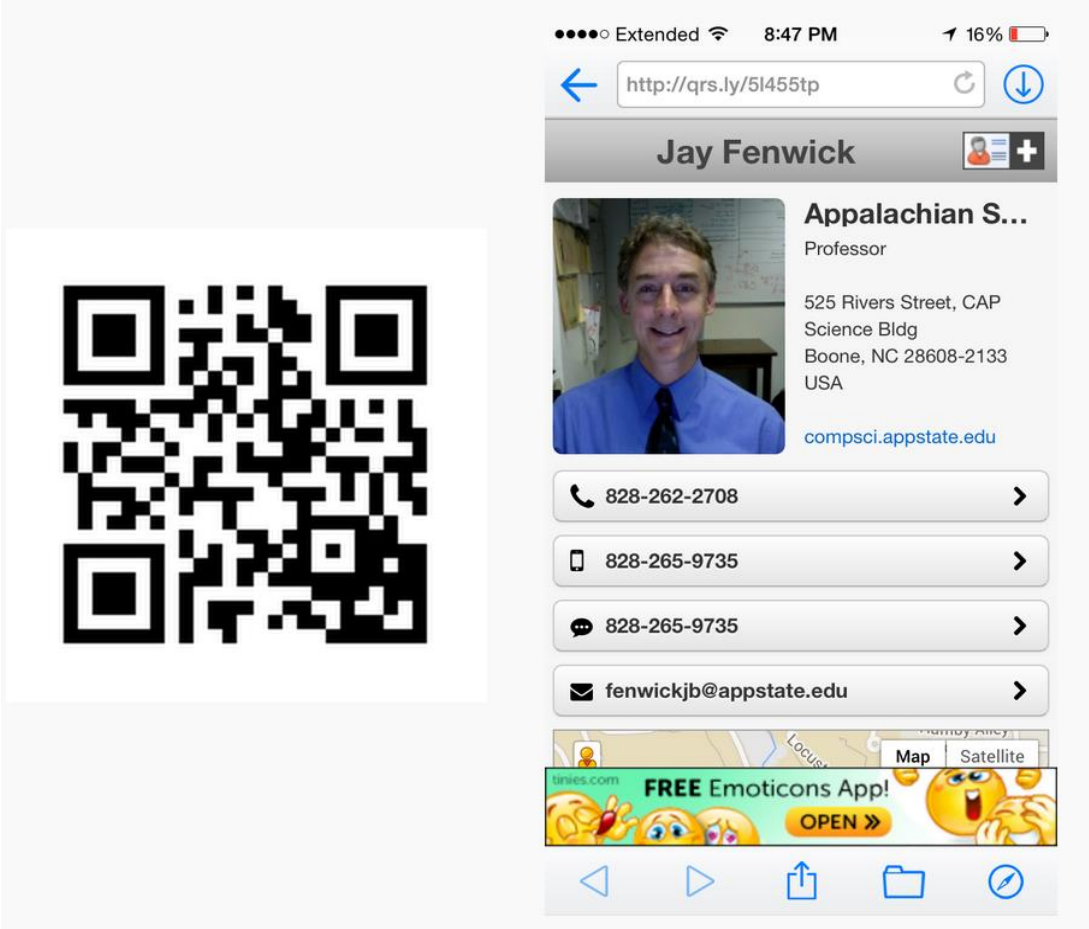


Figure 4.2: QR code and the result of scanning that code

Manual grading of this assignment requires downloading all the QR code images submitted. Opening each image to reveal the unique blocked pattern. Then, using a mobile device equipped with a QR code scanner app to scan the displayed QR code image. The mobile app should display the corresponding student digital business card.

The AGAF tool to grade this assignment uses Microsoft C#.NET because of the availability of QR image processing external libraries targeting that development environment. This C#.NET project consists of four classes that encapsulate six methods. In addition, fourteen C# libraries are utilized as well as three external libraries, two from ThoughtWorks and one from HtmlAgilityPack. This AGAF tool has an input that is the

folder of all the submitted student QR code image files. Figure 4.3 diagrams the program flow consisting of seven stages. The grading tool first processes each image file in this folder by first opening the QR code image. Using the ThoughtWorks QRcode library [3], the encoded URL is extracted from the image. The URL is then used to form an HTTP request for the resulting HTML web page, which is supposed to render as the digital business card. The HTML file is also parsed to ensure that the digital business card HTML contains the required elements, such as an image. This parsing involves using C# objects from the HtmlAgilityPack external library.

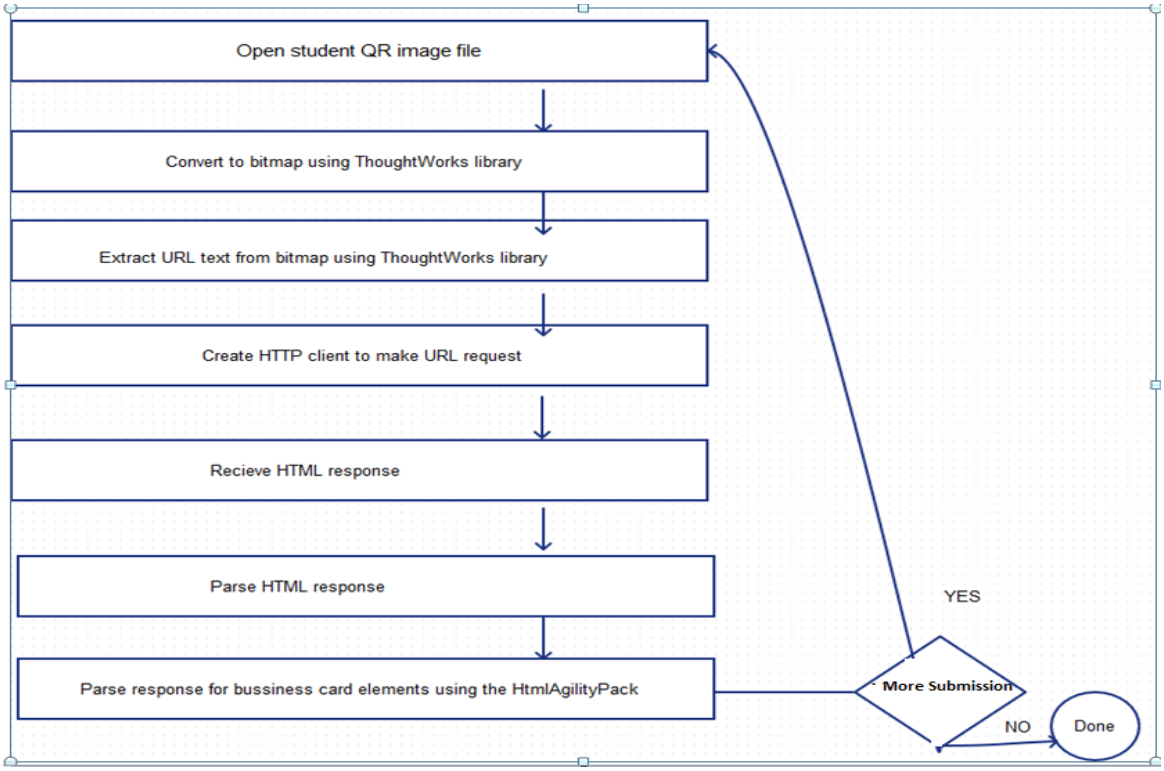


Figure 4.3: QR code grader tool flow diagram

The AGAF tool uses a C#.NET library to display a thumbnail version of the resulting HTML allowing for quick review of the digital business cards by the grader. Figure 4.4 shows a screenshot of this AGAF tool in operation.

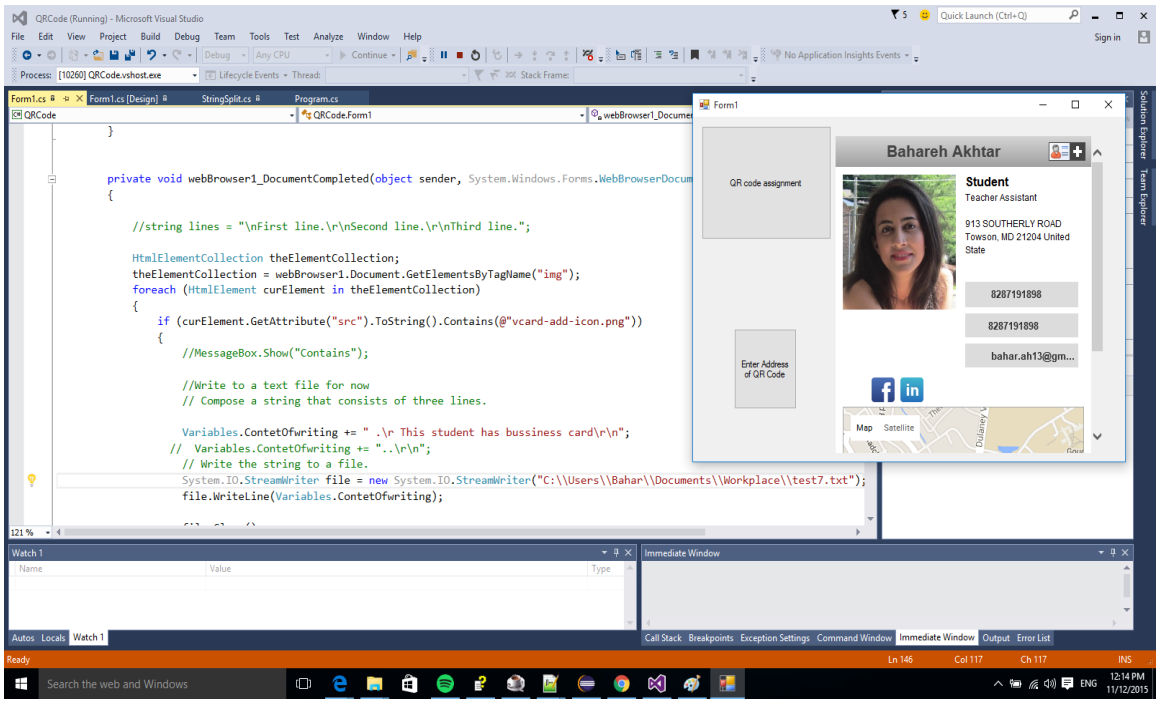


Figure 4.4: A screenshot of the QR code grading tool

There are many external libraries available for the Microsoft .NET development environment. Experimenting with various options to identify the best one for assisting with this task was challenging. Another challenge was parsing the digital business card HTML response to identify the required business card elements. Currently, the student's business card must contain an image attribute.

4.2.2 SurveyMonkey

Another CS1410 assignment requires students to create a simple survey using the popular survey tool SurveyMonkey [23]. Students submit the URL of their finished survey as online text.

The assignment has several requirements that must be satisfied by the student's survey, including:

- 7 or 8 questions,
- At least 4 different types of questions (multiple choice, rating, short answer, etc.),
- One question must be a "multiple choice (only one answer)" type, and
- One question must be a "require" answer.

The context of the survey is left to the student's creativity. Some are about college, others are about sports, etc.

The AGAF tool for grading this assignment also uses the "page scraping" approach. The tool grabs the URL from the student submission and creates an HTTP request for the web page associated with that URL. The response HTML is then processed to identify the required elements. An obstacle of web scraping, described in more detail in Chapter 2, is the volatile nature of the response HTML. For the design of the AGAF survey page scraper, a goal was established to strive for a flexible algorithm. Specifically, we aimed for a page scraping tool that is:

1. Tolerant towards changes in HTML response format from SurveyMonkey
2. Able to parse HTML files to identify the varied requirements.

As described more fully in Chapter 2, an HTML response is structured. It contains a document element, as well as block, inline and interactive elements. Figure 4.5 shows some of the text of an HTML response from SurveyMonkey.

```
1 <!DOCTYPE html >
2  <head>
3    <title>Students Of Appalachian State Survey </title>
5  </head>
6  <div
7    class="questions clearfix">
8  <div data-qunumber=”1”
9          class=”question-datetime”>
10  <div data-qunumber=”2”
11          class=”question-singleanswer required”>
```

Figure 4.5: Partial HTML of a survey

There is an alternative way to represent the HTML document structure that is called the Document Object Model (DOM). Figure 4.6 depicts a DOM representation of the HTML response in Figure 4.5. The DOM representation is used by external tools and libraries for processing HTML text including the jsoup library used by AGAF.

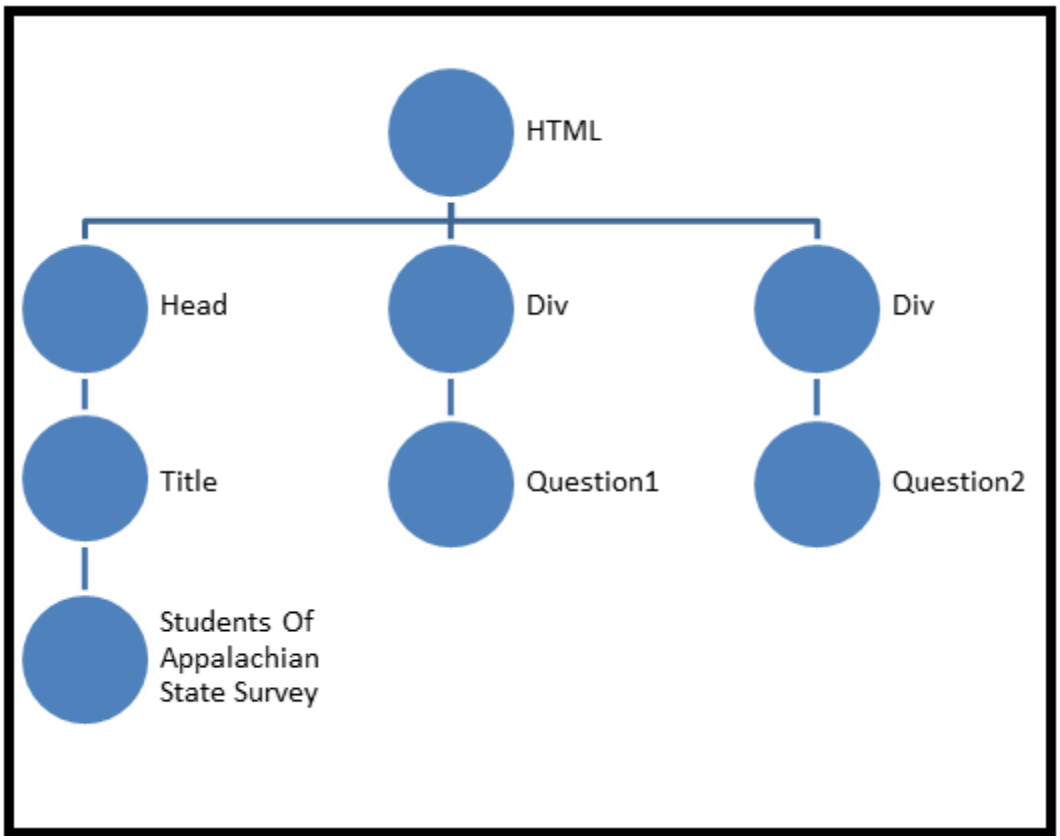


Figure 4.6: HTML document model of SurveyMonkey website

This AGAF tool is written in Java and uses the JSOUP library to assist with the web scraping tasks. This Java project consists of two classes that encapsulate five methods totaling over 500 lines of Java program code. In addition twelve Java libraries are included as well as the jsoup external library [11].

The tool uses a series of regular expressions to extract desired information from the HTML text. The number of questions is determined by scanning for occurrences of the “data-qnumber” attribute within the “div” section that defines a survey question. Similarly, these tags can be scanned to learn about the question types and other question attributes including if the question is required or not.

Figure 4.7 diagrams the program flow followed by this application. Stage 2 illustrates the special task of using a regular expression to extract the student's URL from the submission text. It turned out that students submit more than just a URL. They often put the URL in a sentence, such as

`"My survey is located at http://surveymonkey.com/s/SJKBZA."`

Consequently, the URL must be identified within the submission text and extracted for further use by AGAF.

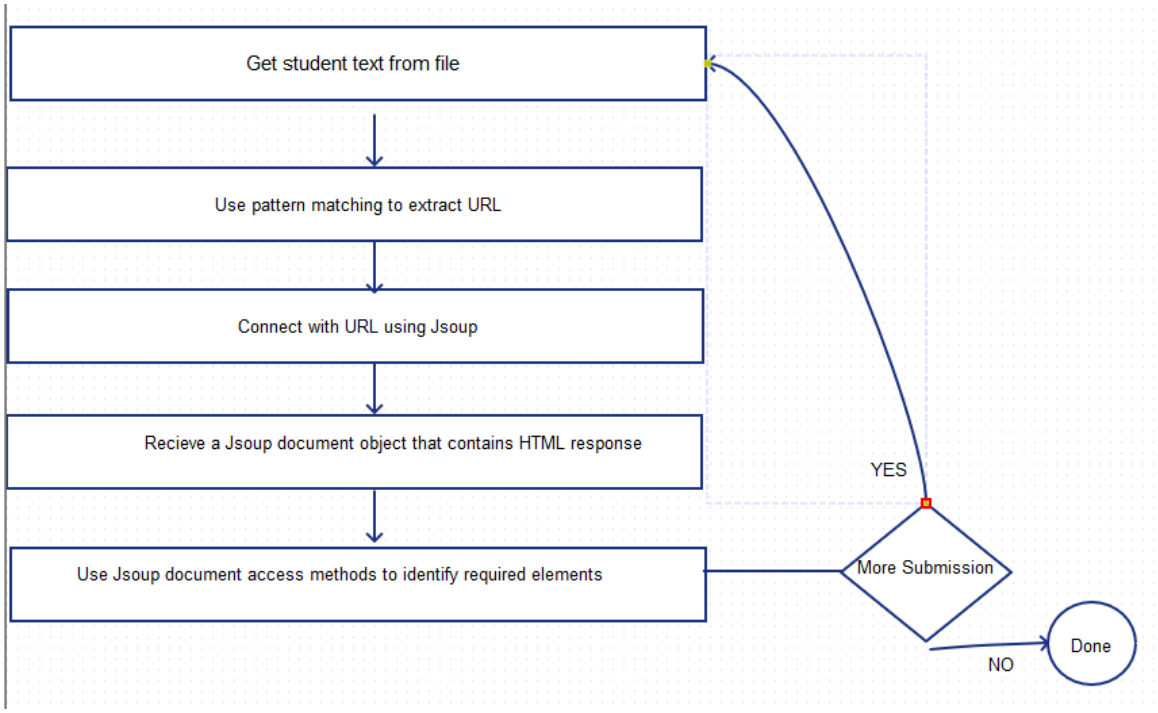


Figure 4.7: Flow diagram of AGAF SurveyMonkey grader

The tool works well on surveys that use only a single page. Multiple-page surveys are not currently supported. The problem lies in decoding the URL of subsequent pages. Because a series of regular expressions is used, automated feedback can be quite precise about required elements that are not discovered.

4.2.3 Blog posting

Another CS1410 assignment requires students to create a blog. Students submit the URL of their finished blog as an online text. The main requirement of the assignment, after creating the blog page itself, is making two posts. The content of the blog is left to the student's creativity. Thus, they post about whatever they want.

The AGAF tool for grading this assignment also uses the "page scraping" approach as diagramed in Figure 4.8. The tool grabs the URL from the student submission and creates an HTTP request for the web page associated with that URL. The response HTML is then processed to identify the required elements. The AGAF blog scraper tool needs to scan the HTML for specific elements, so we needed to figure out the structure of these pages.

This AGAF tool is written in Java and uses the jsoup library to assist with the web scraping tasks. This Java project consists of two classes that encapsulate eight methods totaling about 400 lines of Java program code. In addition, twenty-one Java libraries are used. The last stage is the part that identifies the required elements unique to this specific assignment. At this point, the tool uses a series of regular expressions to extract desired information from the HTML text. Figure 4.9 shows an example of the structured HTML response. The number of published posts is determined by scanning for occurrences of the "status-publish" attribute inside of the "article" document element. Automated feedback can be quite precise about missing required elements.

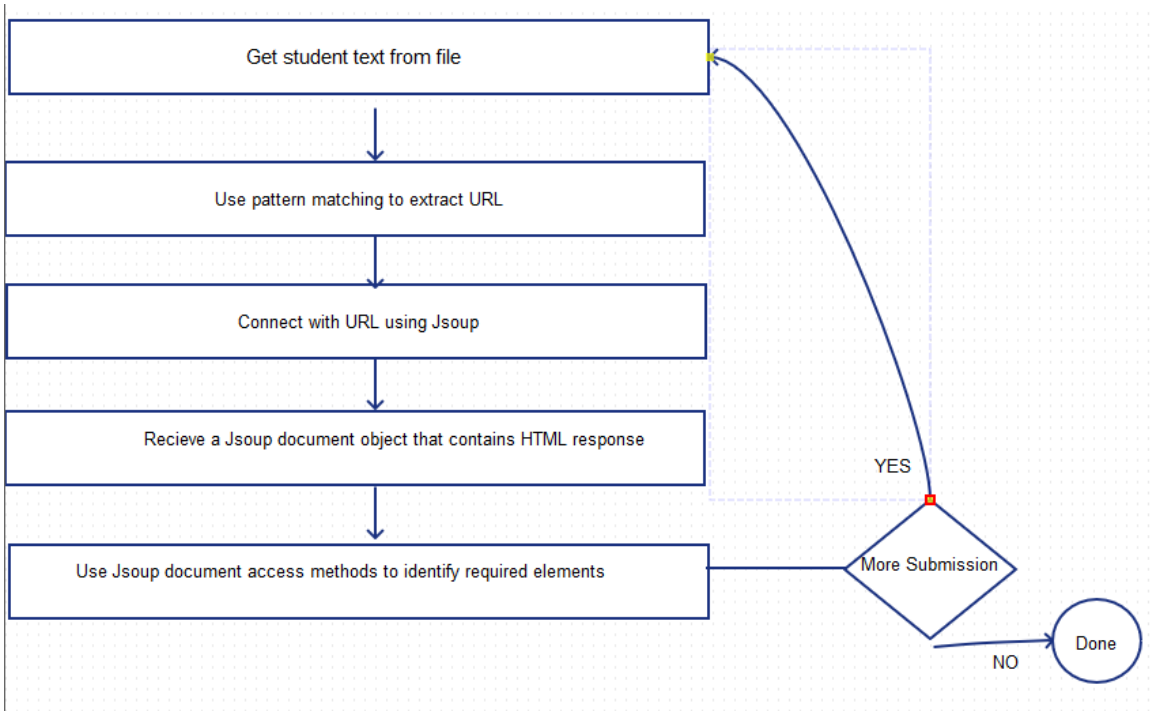


Figure 4.8: Flow diagram of the AGAF blog grader

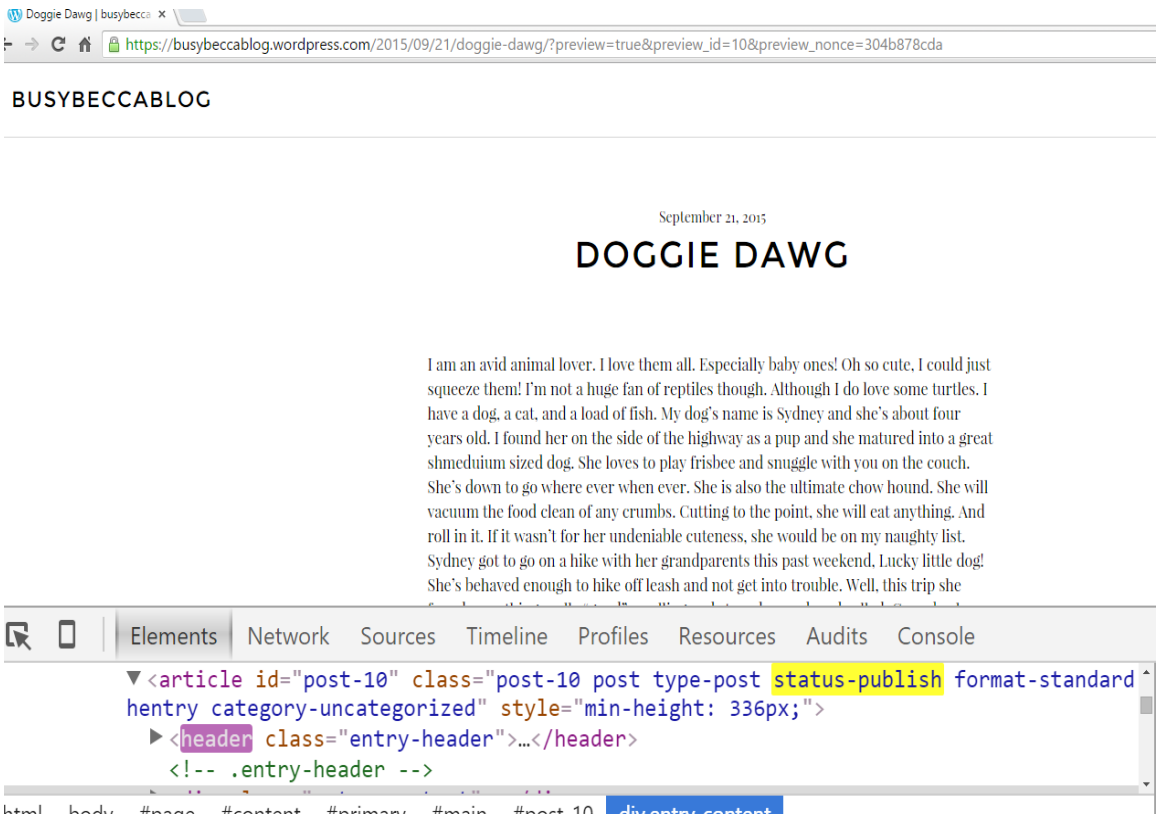


Figure 4.9: HTML of a blog

4.3 Email Processing

Another CS1410 assignment has students create a free account with the professional networking service LinkedIn. Students must create a public profile page and use the "connections" feature of LinkedIn to "connect" with the instructor's LinkedIn account. Students will upload the URL of their public profile page.

Manual grading for this assignment involves two steps. The first is to ensure that students created a public profile page. This involves copying the submitted URL into a browser address bar and seeing that a page is returned and rendered. If a student submits an incorrect URL then the notorious "Error 404 Page Not Found" is rendered. To verify that students did the "connection" requirement, the instructor checks the email account associated with their LinkedIn account. LinkedIn sends an invitation email to the prospective connection when a connection request is sent. Consequently, the instructor's email account has invitation emails from LinkedIn that identify the name of the person requesting the connection, who in this case is the student. The instructor can quickly browse these email subject lines to grade the connection request requirement.

Automating the grading for this assignment, an AGAF tool was written in Java to grab the student's URL submission for their LinkedIn public profile page. AGAF then makes an HTTP request to the server identified in the URL. If the HTML response is valid, meaning that it is not a "Error 404 Page Not Found" response, then the student is credited. The second part requires processing email records. The AGAF tool assumes that a file is available as an additional input. This file contains all the invitation emails and it is then processed to identify a connection invitation email from the student. Most email clients provide a means of exporting email messages into a file for download.

Figure 4.10 diagrams the program flow followed by this Email processing application.

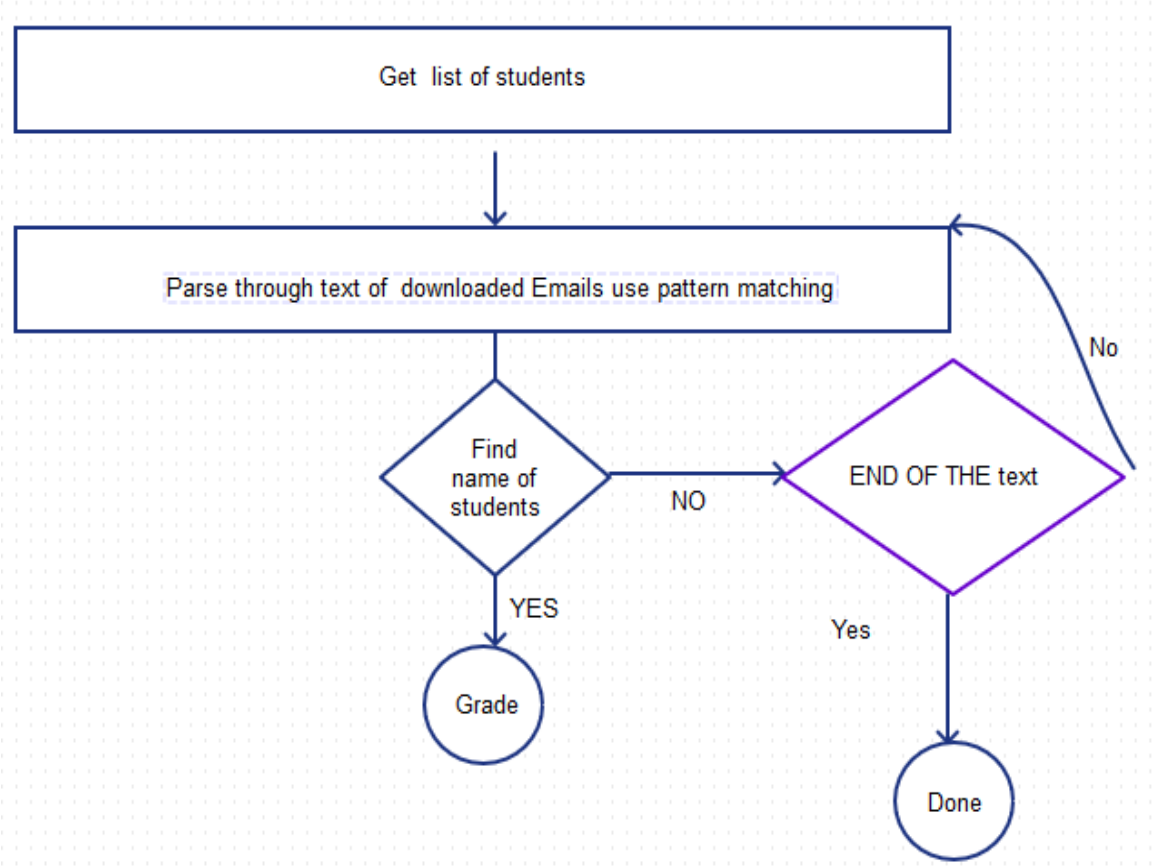
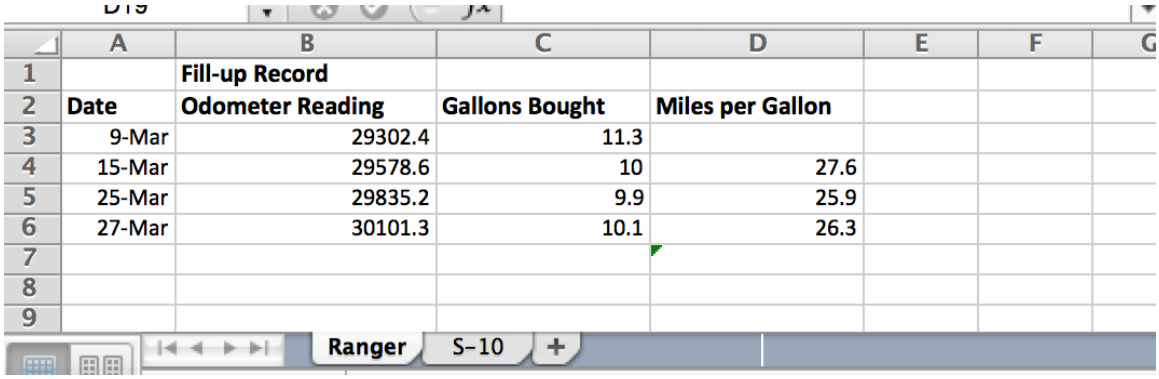


Figure 4.10: Flow diagram of the AGAF Email processing grader

This was a relatively straightforward tool to add to the AGAF tool suite. This Java project consists of two classes that encapsulate five methods totaling nearly 350 lines of Java program code. In addition thirty Java libraries are used. One issue that sometimes appears is that LinkedIn seems to not always generate a connection invitation email. In these cases, students are graded with a 0 for this part. Thus, currently, the instructor must double check these cases manually, but the volume of manual grading has been much reduced.

4.4 Excel Processing

The CS1410 course has several Excel assignments that cover a wide variety of spreadsheet features. Students are required to use multiple worksheets in a workbook and to rename the worksheets. Students are required to "style" column headings, for example to bold text or use a larger font or use background shading. The most important aspects however, are the use of functions (such as IF or AVERAGE) and formulas to perform calculations.



	A	B	C	D	E	F	G
1		**Fill-up Record**					
2	**Date**	**Odometer Reading**	**Gallons Bought**	**Miles per Gallon**			
3	9-Mar	29302.4	11.3				
4	15-Mar	29578.6	10	27.6			
5	25-Mar	29835.2	9.9	25.9			
6	27-Mar	30101.3	10.1	26.3			
7							
8							
9							

Ranger S-10 +

Figure 4.11: Solution of Excel MPG assignment

Automating grading is complicated due to the fact that there are no predefined values for raw data. For example, one assignment has students develop a simple Miles Per Gallon (MPG) workbook to track vehicle mileage and fuel consumption. Figure 4.11 shows a typical student submission. The wording of the column headings is not mandated so it varies from student to student. Similarly for the worksheet names and the particular styling effects. Also, the raw data is not mandated; students are allowed to use "sensible" values for odometer readings and purchased gallons of fuel. Even the columns are not mandated, some students may leave column A blank; others may have the gallons column before the mileage column. All of these factors complicate the development of an automated grading tool.

The AGAF tool developed to grade the MPG Excel assignment is written in C#.NET. In addition, an external tool was used that aids in developing complex regular expressions to be used in the C# program. This C#.NET project consists of five classes that encapsulate eight methods totaling almost 550 lines of C# program code. Also, six C# libraries are used as well as one (Interop) external library. The C#.NET development environment provides an "Interop" library that exposes an application programming interface (API) to programmatically interact with Microsoft Office tools and documents. Using the Interop API, the AGAF grading tool can refer to workbooks, worksheets, and spreadsheet cells. Moreover, the tool is able to identify the various stylings that a student may employ for their column headings.

To help verify the formulas inside of cells, the Expresso professional regular expression development tool [7] is used. Figure 4.12 shows a screen capture of the Expresso tool being used to develop a regular expression for the IF function that students are required to use for the MPG assignment (the values in column D of Figure 4.11 are computed using an IF function).

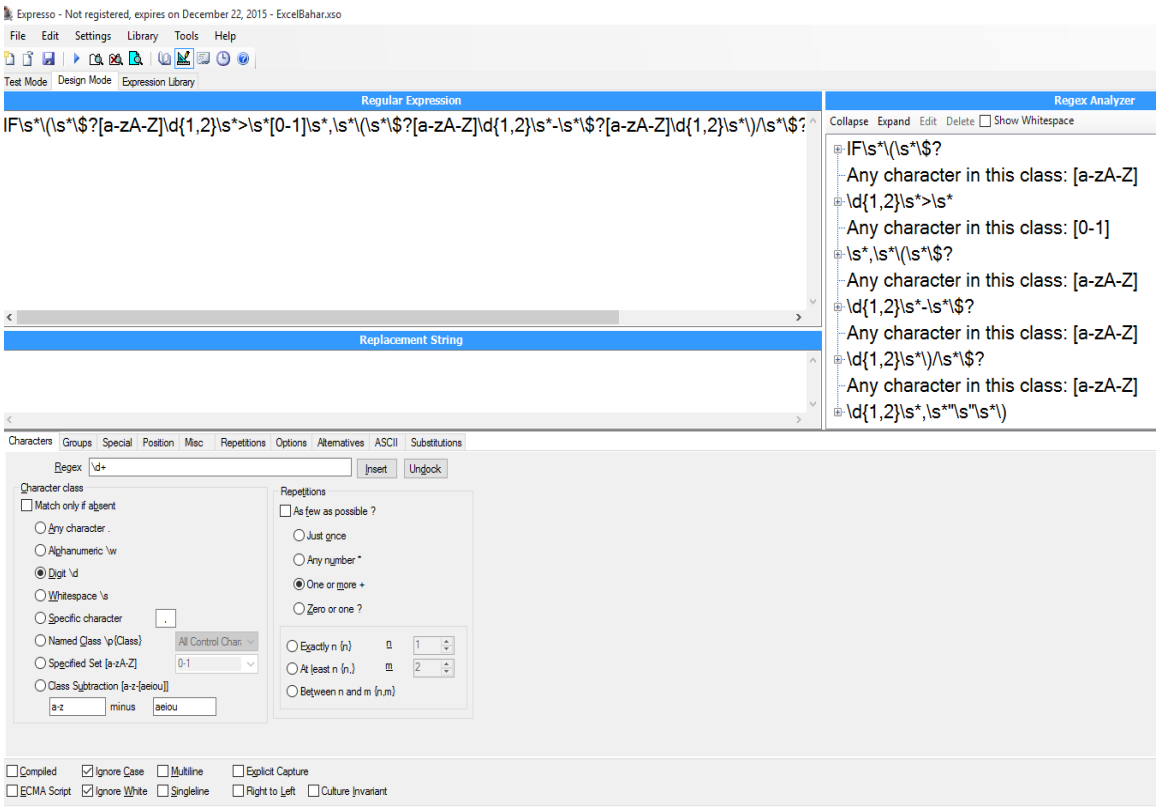


Figure 4.12: Expresso tool screen capture

The use of a regular expression allows for student flexibility in forming their formulas and function expressions. For example, several student submissions are shown below for computing the mile per gallon value in column D. Notice the differences in cells, parentheses, spacing, etc. The regular expressions used by AGAF deemed all these functions as correct. It should be clear that these are complex expressions.

```
IF(D24>0,(C24-C23)/D24," ")

IF(C13>0,(B13-B12)/C13, " "  )

IF(C9>0,(B9-B8)/C9,"")

IF($C9>0,($B9-$B8)/$C9," ")
```

While a regular expression can verify the correct structure of a formula, such as the IF function above, it does not necessarily verify that the correct values are being calculated. For example, the student could erroneously be calculating gallons per mile. However, we can make some inferences based on the cells used in the formulas and the data values used. For example, computing miles per gallon means dividing by gallons. The regular expression expected a division operator and this has already been verified, so we know that the student's formula contains a division and we can extract the cell reference that is the divisor. We can then use the Interop API to examine the value stored in that divisor cell. Since this is the cell that contains the gallons value, this value can be checked to make sure it is realistic; for example that the value is greater than zero and less than 50. We can also look for the column heading of this cell to see if it matches a regular expression for the word "gallon."

Another feature of the automated Excel grading tool is a check for possible plagiarism. The Microsoft Office applications are installed on machines and use the name of the registered machine owner/user. When a document is created, this owner/user information is saved into the file along with creation and modification timestamps and other information. We can extract this information to compare to the student submission being graded or previous student submissions. This is not a foolproof check, but can alert an instructor to a situation that may require more thorough investigation.

Figure 4.13 shows a flow diagram of the Excel AGAF grader. As Figure 4.13 shows, the grader has seven stages for each of the Excel submissions.

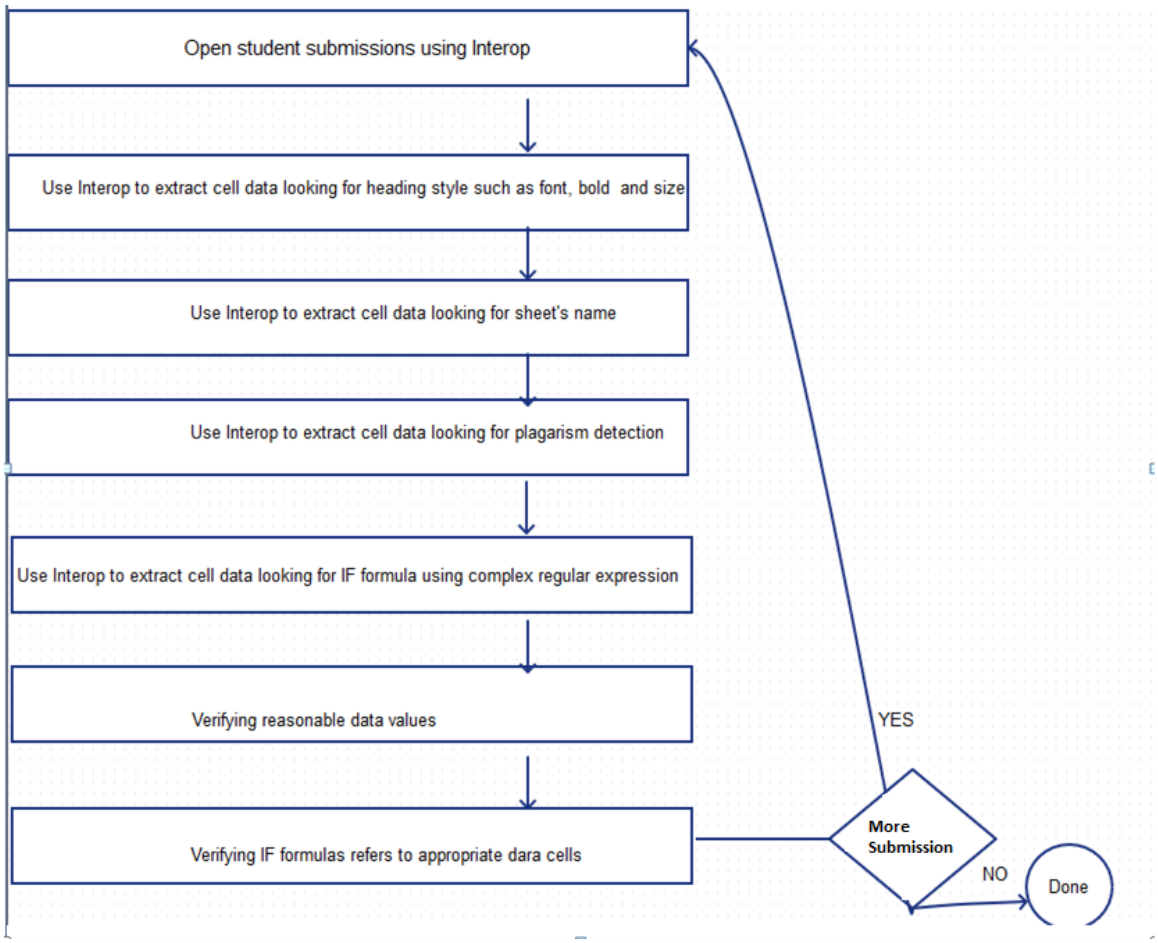


Figure 4.13: Flow diagram of the Excel MPG grader

Figure 4.14 shows a fragment of the output generated by this AGAF tool. We can see the filename of the submitted work (recall that the student name is embedded in the filename along with the assignment name). Also displayed is the author name embedded into the file by the Microsoft Office application. The formulas used by the students are also shown, which has proved useful because they sometimes design overly complicated formulas that may actually work. The sheet name of the excel file and a final grade of submission based on a defined rubric is shown. A descriptive rubric file is also generated. This can be seen on the right side of Figure 4.15.

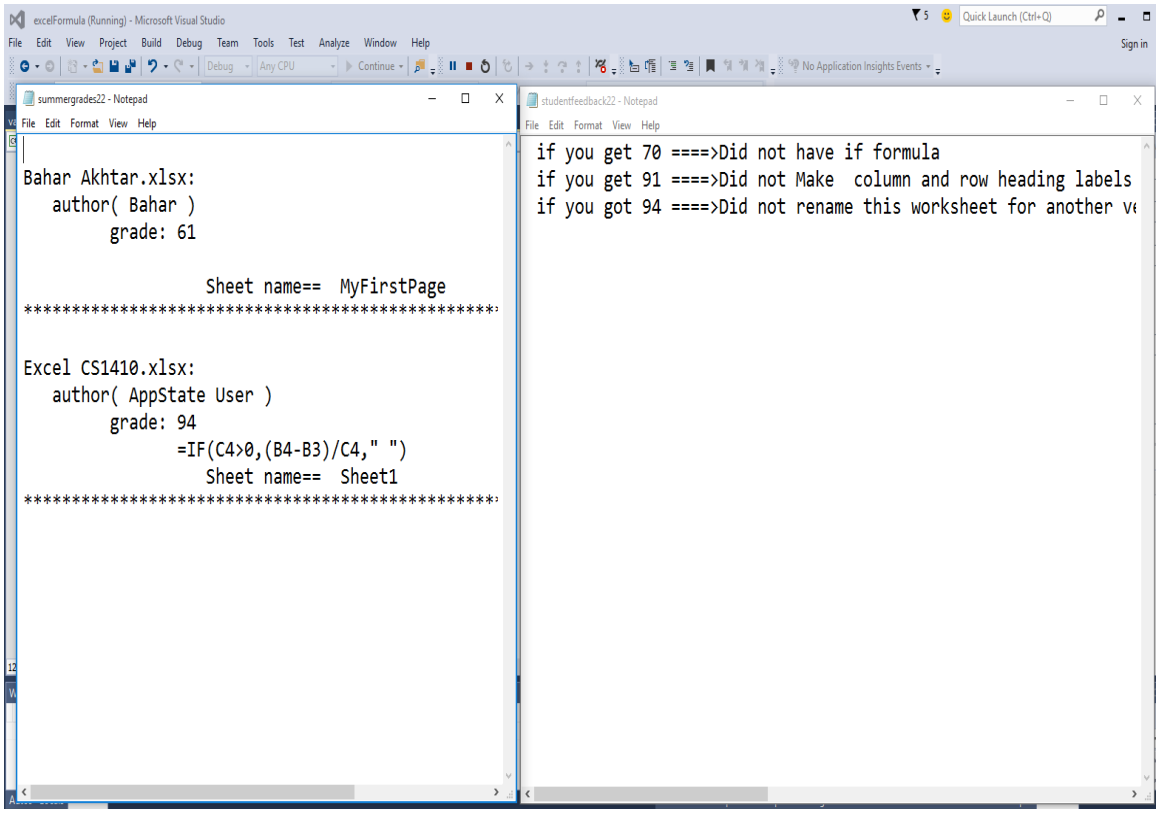


Figure 4.15: Example screen capture of the detailed output and feedback

4.5 Image Processing

Another CS1410 assignment has students complete several lessons published by the web-based skill development company Codecademy [4]. Codecademy has numerous courses covering a variety of modern web development skill sets. This assignment has students create a free Codecademy account and use the website to complete several modules of their Python course. The Codecademy account includes a course overview page that lists all the Python course modules and puts a green check beside the module when they complete it. Students upload a screenshot of this module checkmark to show they completed the module. Figure 4.16 below shows a student submission for this assignment. This particular image shows two lessons being successfully completed.

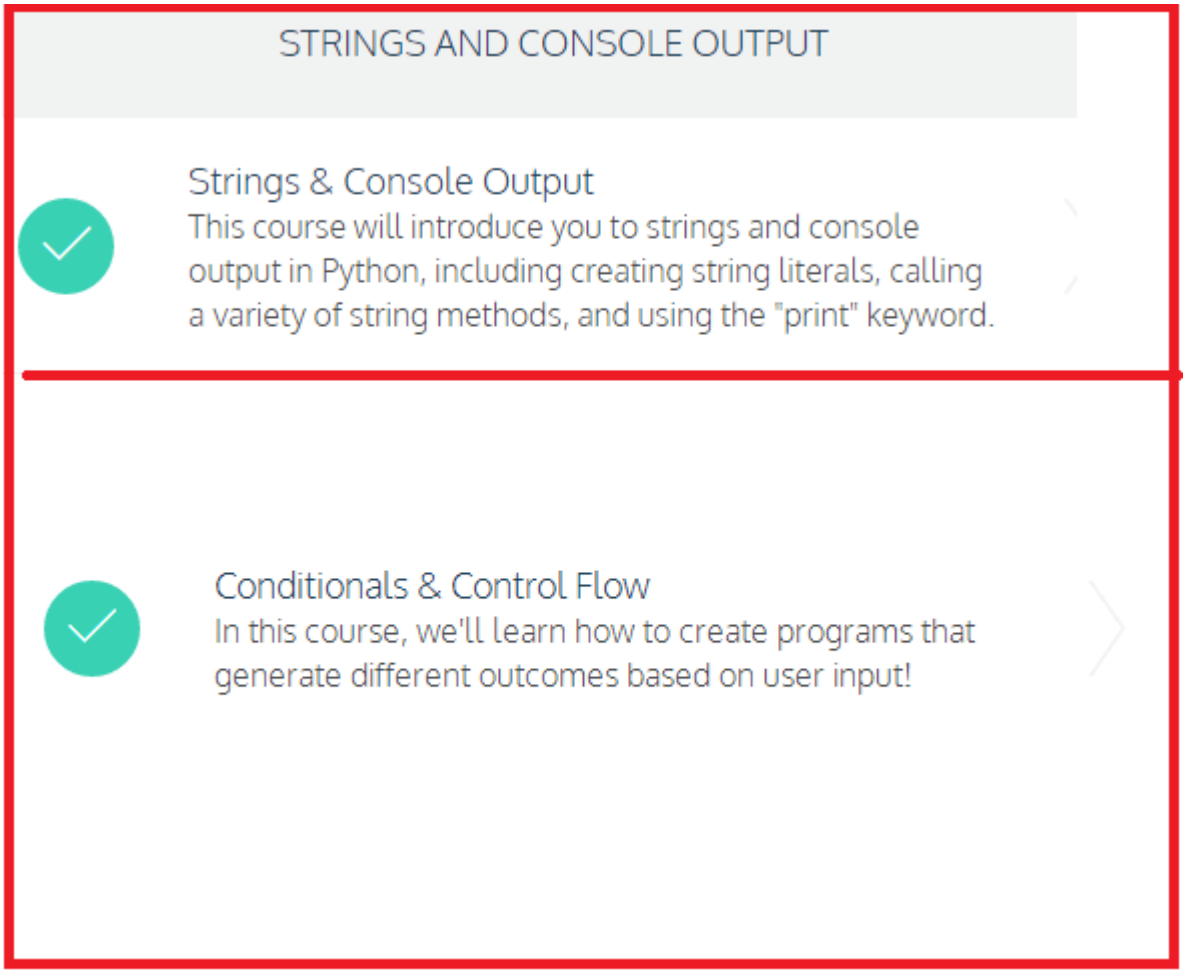


Figure 4.16: Codecademy assignment submission

Manual grading these assignments involves downloading all the student submissions. Each submission is an image file showing a lesson name and a check mark. Students sometimes submit images (like the one in Figure 4.16) that have superfluous image data cropped out. However, others simply submit an image capturing their entire screen contents. A human grader can easily verify the check mark and the lesson name.

Two AGAF grading tools have been developed to support varying requirements. One tool applies image processing techniques to confirm a particular image. For example,

this tool could be used to confirm the existence of the check mark in the green circle. This grading tool is based on computing scale-invariant feature transform (SIFT) key points. The C#.NET development environment provides an external library, OPENCV, that contains SIFT support routines. This method works well for distinct images. While this tool didn't work well in this context since the same check mark image is used for each lesson, similar systems employ a different and unique "badge" for completing lessons. This tool would work well for those kind of assignments.

A second AGAF grading tool for this type of assignment uses Optical Character Recognition (OCR). This approach for grading image submissions has the following features:

- Scale invariance meaning that image size can vary,
- Can extract texts in images even with poor quality,
- Uses a free, open source resource to access the OCR library [28],
- Integrates Microsoft OneNote OCR to read and parse text in the images, and
- Capable of supporting a wide variety of similar assignments.

Microsoft OneNote [17] is a standalone application often delivered with the Microsoft Office suite of applications that has the ability of storing notes, drawings and images. In particular, OneNote contains an OCR library that can extract text from images.

Figure 4.17 diagrams the program flow followed by this application. The grading tool imports each student submission image into a OneNote page. Then the OneNote OCR library code is used to parse through the internal OneNote document and extract the text. The text should match the expectations for the associated Codecademy lesson. This technique is borrowed from an open source project by Matthias Welz [28] and adapted

here as a stage of our automated grading workflow. This C#.NET project consists of seven classes that encapsulate twenty methods totaling nearly 450 lines of C# program code. In addition thirty-nine C# libraries are included as well as two external libraries, Xml.Linq and Microsoft.Office.Interop.OneNote.

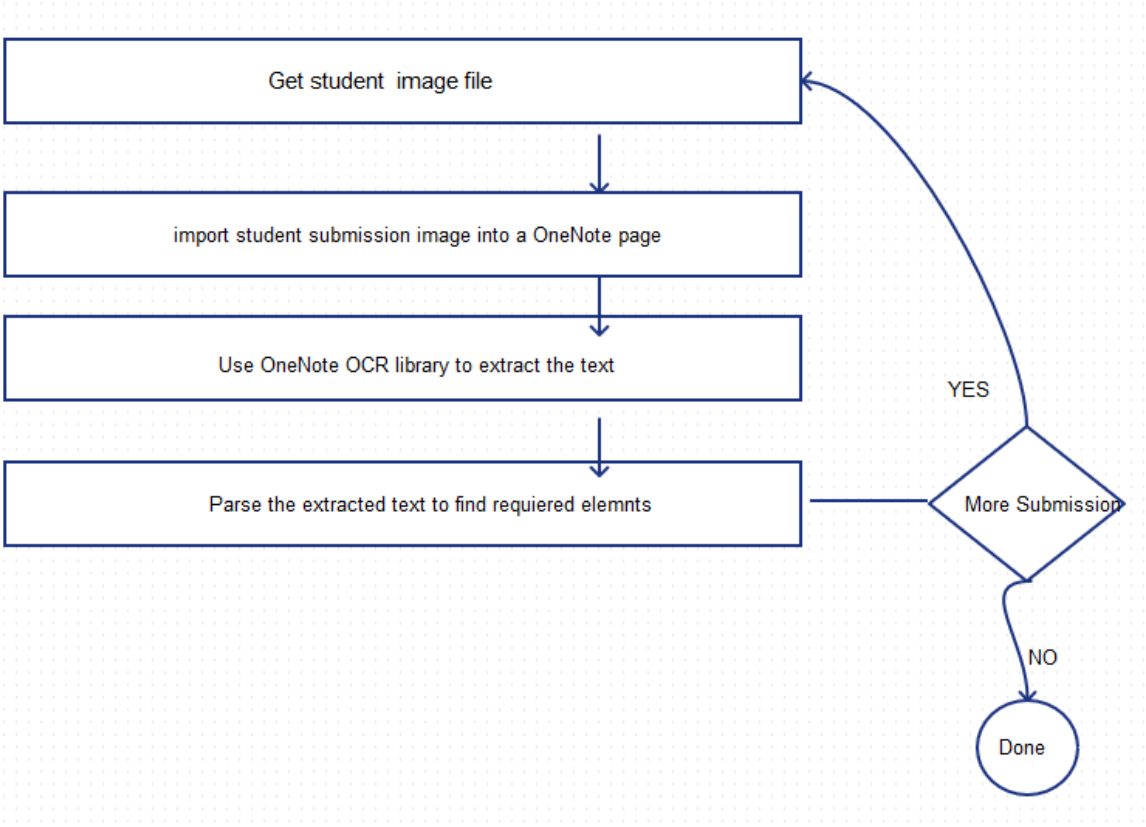


Figure 4.17: Flow diagram of the AGAF Codecademy grader

There were several challenges in developing this AGAF tool. It took a number of experimental variations to isolate the best set of configuration parameters for this multi-stage workflow. But trickier challenges dealt with memory management and thread management. It is relatively straightforward to create a OneNote page for each student submission. But when grading class sizes as large as is needed for CS1410, memory

usage became a problem. Investigation into the problem uncovered that each conversion of an image into an OneNote page creates its own user process. Eventually, the memory and resources are impacted. A solution to this problem was to avoid creating a new process each time. Instead a process "pool" was used. If the pool contains an available process then that already created process can be re-used for the new task of converting an image to an OneNote page. Another challenge of this research dealt with thread management. A grade of 0 was computed for some student submissions when the images they submitted were acceptable by manual inspection. Investigation of these erroneous results led to discovering a poll interval value. When a process was re-used to convert a student's submitted image into an OneNote page, a delay is used to await the completion of the conversion process. Extending this delay interval from 250 milliseconds to 1000 milliseconds provided sufficient time for the conversion.

This application also provides detailed feedback. The output is a text file containing the names of students and their computed assignment grade, but also shows the text extracted from their images in addition to the expected text. This AGAF tool also provides a more interactive mode of inspecting student submissions. Instructors can "slide view" the images. While manual, this still saves much time over manually opening and closing each image file.

Chapter 5 - Evaluation

This chapter evaluates the efficacy of the AGAF Automated Grading And Feedback System for the Summer 2015 offering of the CS1410 course. The summer enrollment of 23 students is substantially lower than a regular term thus allowing assignments to be graded using the AGAF tool suite and manually as well. The manual grading was done by the instructor during the term. The AGAF grading was done independently of the instructor and the term. We will primarily consider two measurement criteria: time spent grading and a measure of similarity with manual grading. The first metric of the time spent grading is pertinent because computer literacy courses often have large enrollments so time is an important practical criteria. In terms of how similar the automated grades are to manual grading, it might seem that the automated grades should be identical to the manual grading. While that is often the case, we do demonstrate cases where the instructor made mistakes. Thus, a 100% match is not necessarily the goal and differences are carefully analyzed.

5.1 Excel assignment

Manually grading the Excel MPG assignment requires opening each submission Excel file and visually verifying the various column header styling and the renamed worksheets. Verification of the MPG formula calculation involves a quick visual

inspection of the values displayed but followed by selecting one of those cells to expose the underlying formula. A visual inspection then confirms if students are calculating the correct value with a formula or if they are simply entering text that looks correct. A last step of verification ensures that there are multiple worksheets in the file and by clicking on that second worksheet to confirm the presence of the expected data.

Manually grading the 23 student submissions for this assignment took about 25 minutes, averaging about 1 minute per assignment. By contrast, the AGAF tool graded the assignments in about 2 seconds. Extrapolating this performance to a regular semester enrollment of 125 students, the time savings of auto grading with AGAF is about 2 hours! Manual grading would demand about 125 minutes while AGAF would complete the task in less than 1 minute.

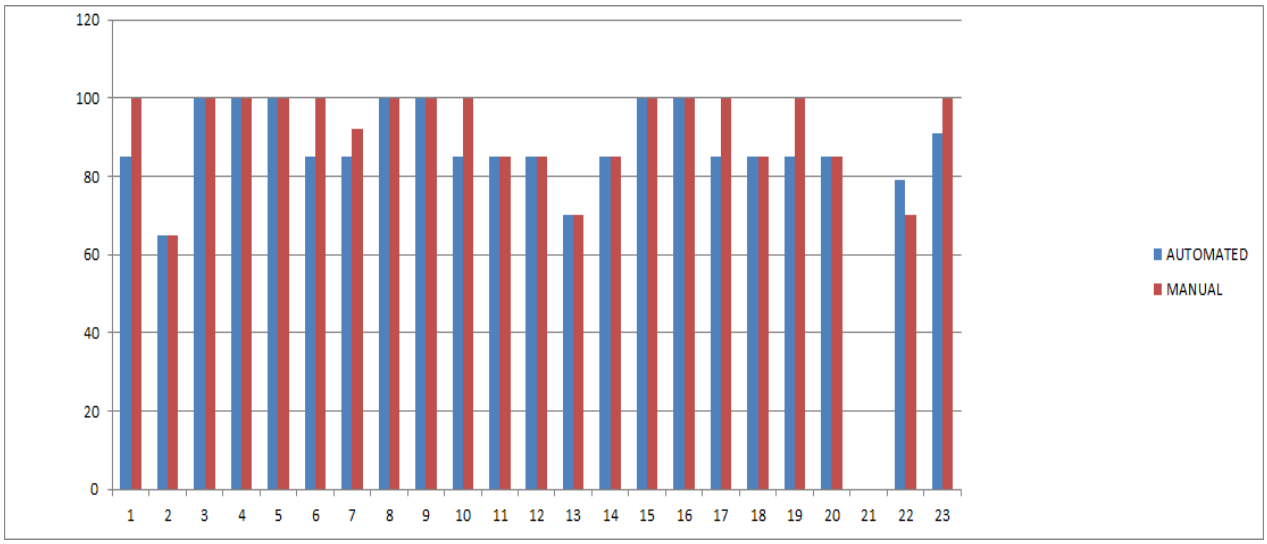


Figure 5.1: Grades of AGAF Excel grader and manual grader

In terms of similarity, the AGAF grading tool agreed with the manual grading in 15 of the 23 cases for an initial agreement of 65.2%. In 4 cases (Students # 6, 10, 17, 19) the manual grade was 100 but AGAF graded these at 85. This was due to the fact that

while the student had some IF formulas correct they still had some that were incorrect. Most likely, the instructor did not noticed these small discrepancies or only happened to look at the cells with correct formulas. This is an example of the error-prone nature of human grading this type of student work! In another 100 manual vs. 85 AGAF case (Student #1) the student's formula looked correct and behaved correctly but only by accident. The formula is supposed to check for a gallons value that is not zero in order to prevent the display of a "divide by 0" error message in the cell. In this case, the gallons value was in cell C4 but the student's formula was: IF(B4>1,(B4-B3)/C4,"") which is correctly dividing by the gallons in C4 but is incorrectly checking the mileage value in cell B4 to make the decision. Again, AGAF has demonstrated a case of human error in grading.

In the case of Student #23 a grade of 100 was given by the instructor but a 91 from the automated AGAF grader. In this case, the student submission did not style the column headings on the first worksheet which AGAF used to deduct 9 points. However, the second worksheet did have styled column headings. Apparently the instructor allowed this to satisfy the styling requirement in spite of the assignment saying to "copy everything in the original worksheet into a second worksheet." AGAF has again demonstrated an inconsistency in the manual grading.

In the case of Student #7, AGAF graded the submission lower because the student used a correct formula that did not match the AGAF formula regular expression. Generally, the formula checks for the presence of a numeric value and finding this computes a formula. Student #7 did the inverse, checking for the lack of a numeric value.

This is when the detailed AGAF grading feedback is useful. An instructor can see the IF formula that failed and visually inspect to override the AGAF grade.

Differences between the AGAF grading and the manual grading were generally due to human error and inconsistency. There was one case of AGAF not recognizing a valid formula which has been incorporated into the next version of the tool. We suggest that instructors scrutinize automated graders if the grade is particularly low. We also suggest that instructors continue to verify the results of automated graders by spot checking a random sample.

5.2 Codecademy assignment

Manual grading of the Codecademy assignment is a little faster than Excel because there is less work involved looking through the opened submission. In this case, the grader can quickly identify the green circle checkmark and the expected lesson name. No additional clicking within the document is needed. Occasionally, a student image submission needs to be resized to affirm the text of the lesson. Manual grading of the 23 student submissions for this assignment took about 20 minutes. The AGAF automated grading tool took only 1 second. Extrapolating this result to a regular term enrollment of 125 students means a grader would require about 100 minutes; whereas, AGAF would complete the task in about 5 seconds.

For the similarity criteria, Figure 5.2 shows that the AGAF results agreed with manual grading results in 22 of the 23 cases. This is a 95.7% similarity in the outcomes. The one case that differed, Student #2, was graded manually with a score of 100 but was graded by AGAF with a 0. The reason for the AGAF grade was due to an image of very poor quality. The optical character recognition did not correctly recognize the text in the

image. Indeed, manual grading had difficulty in this case! While the instructor will need to manually grade these submissions these cases are extremely rare.

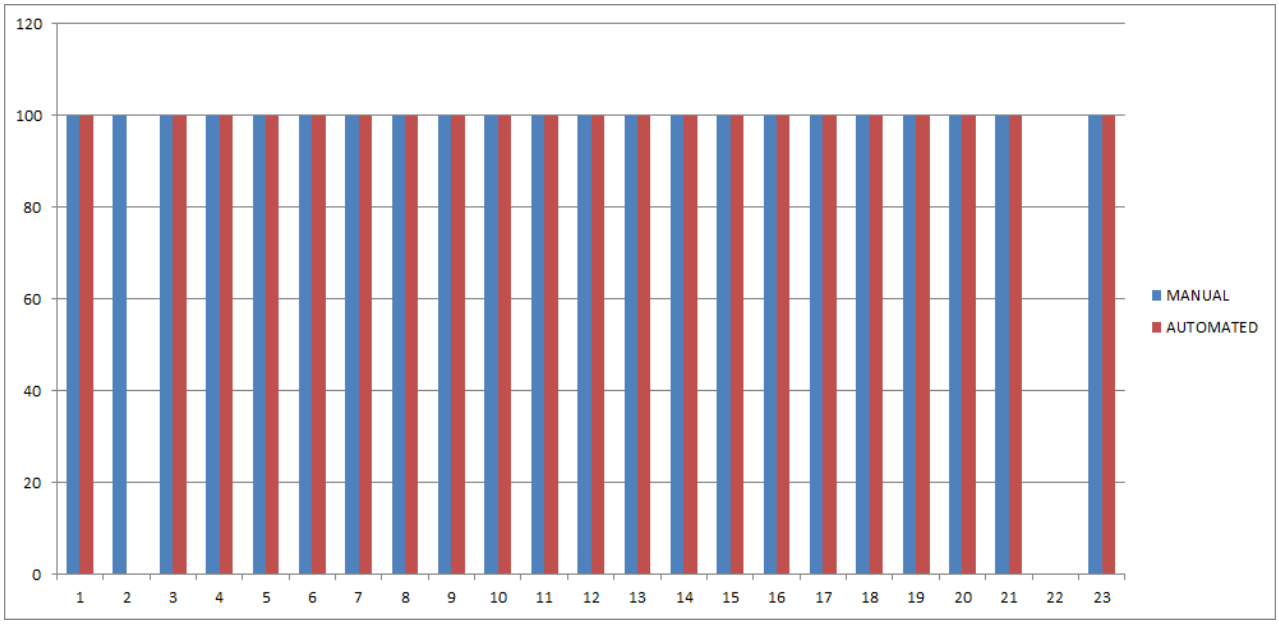


Figure 5.2: Grades of Code Academy grader and manual grader

5.3 Survey assignment

This assignment requires students to build a survey using the Survey Monkey tool and submit the URL of the survey as online text. Manually grading this assignment involves the repetitive actions of opening a text submission file to obtain the URL of the student's survey, pasting this URL into a browser window, waiting for the survey to load, and then verifying the various required elements, such as number and types of questions, etc. Manual grading of the 23 student survey submissions took about 50 minutes. Whereas, the AGAF tool needed only 10 seconds. For a regular term of about 125

students, these values extrapolate to four hours of manual grading compared to 15 seconds for the automated grading tool.

As can be seen in the grading comparison shown in Figure 5.3, the AGAF tool agreed with the manual grade in 18 of the 23 cases for a similarity of 76.3%. In the 5 cases where the grades differed, the student survey had multiple pages. The AGAF tool is unable to identify and inspect this second page. Hence, some required elements are graded as 0 by the automated grader. Manually, the second page link is easily clicked and verified. Knowing this limitation of the automated grader, the instructor can easily update the assignment specification to ensure the student survey only uses a single page.

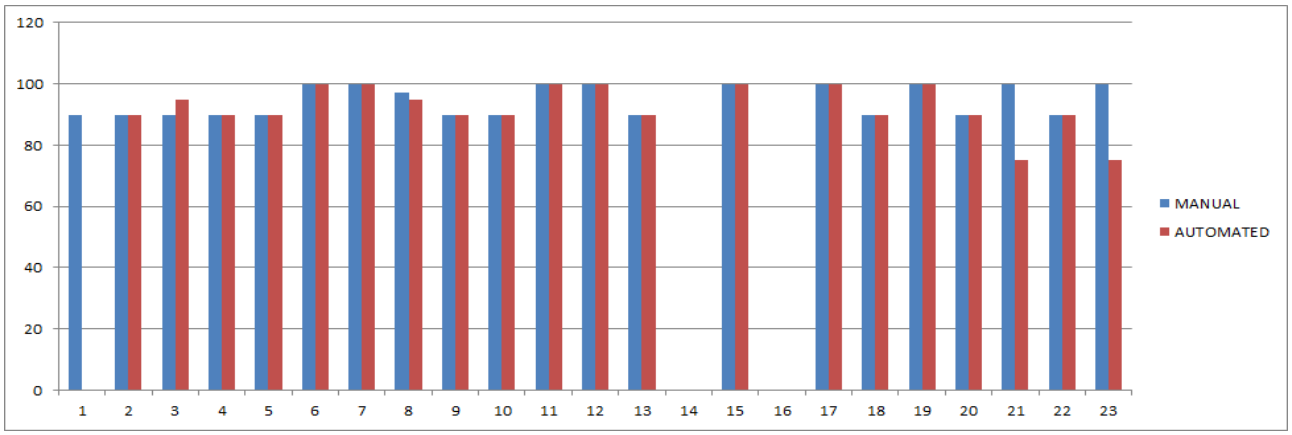


Figure 5.3: Grades of SurveyMonkey grader and manual grades

5.4 Digital business card QR code assignment

This assignment has the student use a web-based tool to construct a digital business card. Their digital business card has a URL, but instead of submitting that

directly, this assignment has the student convert the URL into a 2-dimensional bar code known as a QR code. They upload the QR code image as their submission.

Manually grading this assignment requires opening the submitted image files and for each one using a QR scanner application on a mobile device to scan the opened image file. This causes the digital business card to be displayed on the mobile device where the grader can manually verify the required elements. Manually grading the 23 submissions in this manner took about 30 minutes. The AGAF automated grading tool accomplished the task in 60 seconds. Extrapolating to a regular term enrollment of 125 students, manual grading would take about 2.5 hours compared to 5 minutes for the AGAF grading tool.

As Figure 5.4 shows, the grading similarity in this case was 100% since the AGAF tool grades agreed with the manual grades in each of the 23 cases. To be fair, this is not a complicated assignment for students to complete successfully as is seen in the large number of perfect scores.

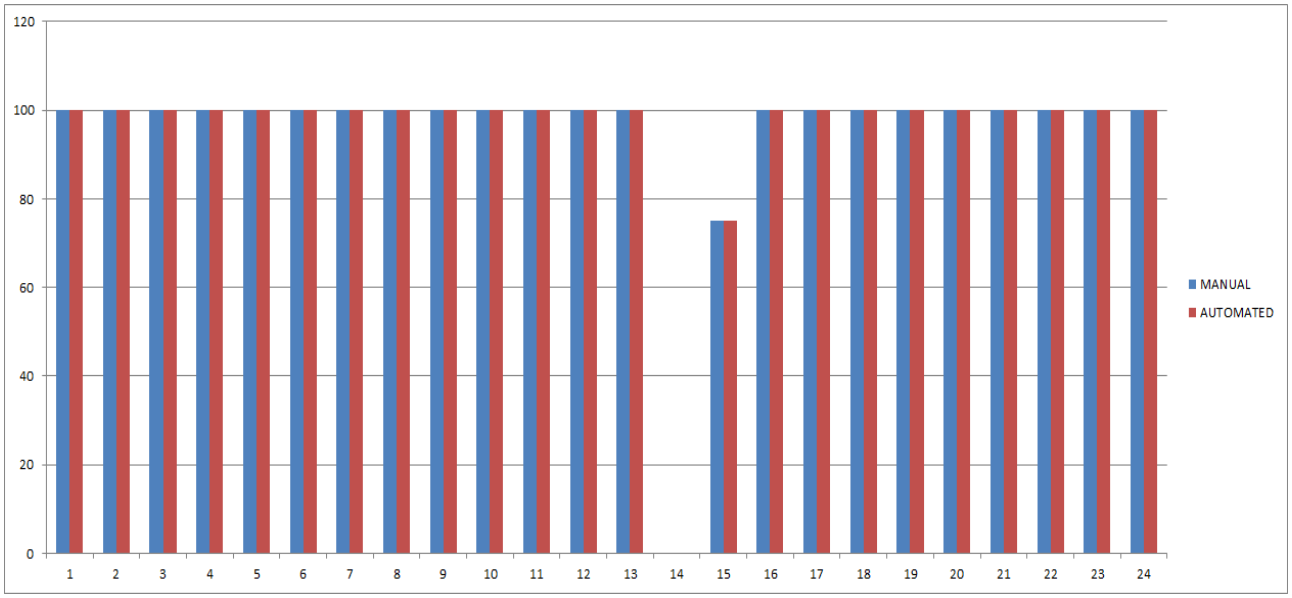


Figure 5.4: Grades of automatic QR code and manual grading

5.5 LinkedIn assignment

This assignment has students create a free social media account at LinkedIn, which is a professional networking service. Students must create a public profile page and request to "connect" with the instructor's LinkedIn account. Students upload the URL of their public profile page as online text.

Manually grading this assignment involves opening the submission text file, copying the submitted URL, and pasting the URL into a browser. If the student's public profile page is displayed then they get credit for this part. To grade the connection attempt, the grader opens the email account associated with the instructor LinkedIn account. Since LinkedIn sends an invitation email when a connection is requested, the grader can search among recent emails for an invitation from a particular student.

The time required to manually grade the 23 submissions was 50 minutes. This time is higher than some of the other assignments because finding an email from a specific student can be a slow endeavor. And this particular problem will probably not scale linearly as the class size increases. The AGAF tool was able to grade this assignment in 5 minutes.

The grading similarity is shown in Figure 5.5. Only 69.6% of the grades (16 of 23) were identical. In all of the differences, the AGAF tool graded at zero because the invitation email was not found. Manual grading in this case of not finding an invitation email involves opening the instructor LinkedIn account and navigating the email client archived emails for connection invitations. It is unknown why some connection requests seem to not generate a corresponding email. Trying to gain further insight into this issue, this AGAF tool was used with the Fall 2015 term assignment submissions. Out of about

70 submissions only 15% of the invitation emails were not found. Some thoughts on improving this component of grading this assignment are described in Chapter 6.

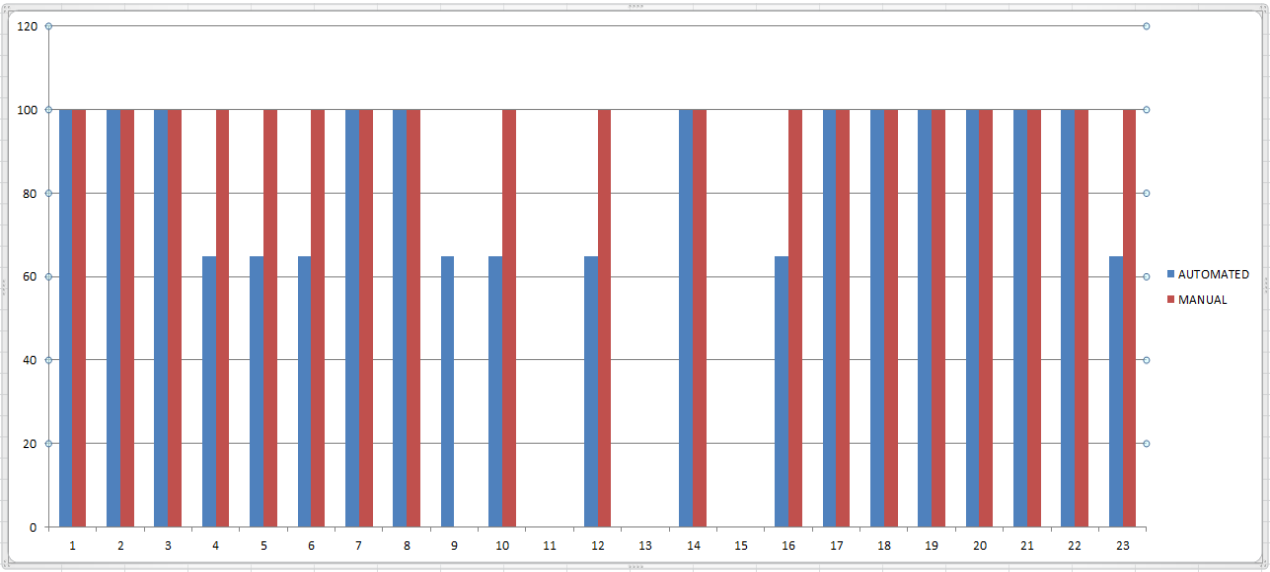


Figure 5.5: LinkedIn automated grades and manual grades

5.6 Blog assignment grades

The blog assignment has students create a blog page using the Word Press blogging web application. In addition, they must post two items to their blog. The content of the blog is left to the student's discretion and creativity. Students submit the URL of their blog as online text.

Manual grading of the blog assignment involves opening each student submission file and copying the URL in the file. The URL is then pasted into a browser window which in turn displays the student's blog site. The grader then visually inspects the site to identify two posts. Manual grading of this assignment for 23 students took about 15 minutes, but the time for the AGAF tool was only 2 seconds. Figure 5.6 shows that the

grading similarity was 100% since the automated grades agreed with the manual grades in all cases. There are not many mitigating scenarios with this assignment, students tend to either get it done completely and correctly or not submitting anything at all. The time savings offered by the automated grader are extremely helpful in these cases.

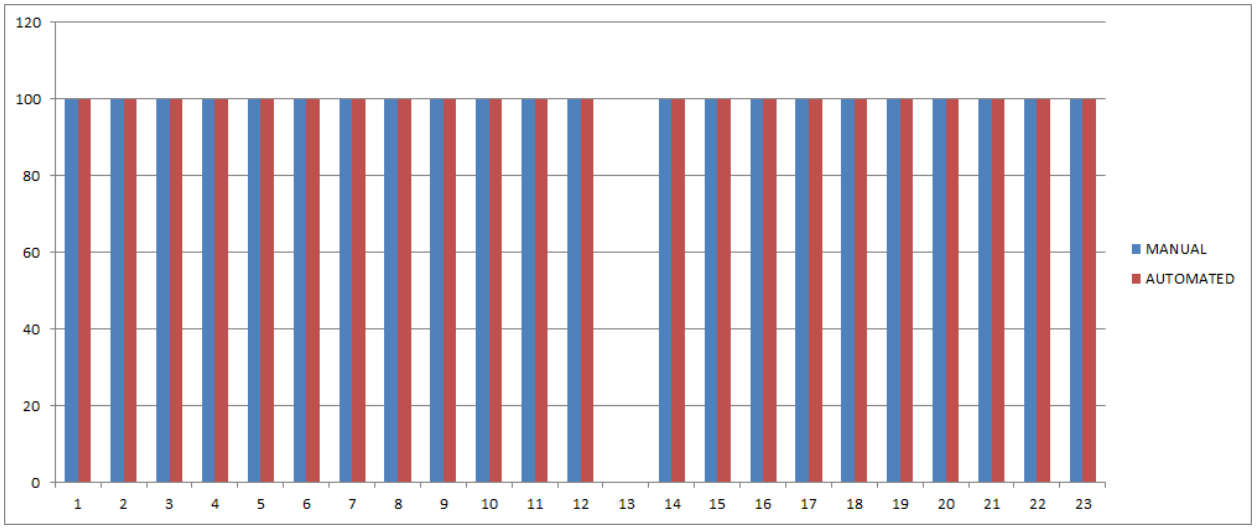


Figure 5.6: Comparison of grades of blog assignment

Chapter 6 - Conclusions and Future Work

6.1 Conclusions

Grading is a human activity thus potentially suffering from human errors and inconsistencies. These issues become more pronounced as course enrollments increase. The introduction proposed that automated grading of a variety of assignments typical to college computer literacy courses could be accomplished. The remainder of this thesis described the development of a suite of automated grading tools named the Automated Grading And Feedback (AGAF) System. AGAF was developed specifically for the CS1410 computer literacy course at Appalachian State University but is flexible enough to support similar courses at other institutions. AGAF contains automated grading tools for a variety of assignment types, including Excel workbook submissions, image submissions (e.g., badge identification), and web page inspection of URL submissions (e.g., surveys, websites).

The AGAF tools integrate numerous technologies to accomplish this variety of tasks. Both the Java and C#.NET programming languages are used. In addition, a number of external libraries are employed to assist with very specific tasks, including the jsoup library for regular expression handling and the OPENCV library for image processing. AGAF tool output contains computed grades but also textual feedback about missing rubric elements contributing to deductions.

The AGAF tools were applied to the summer 2015 offering of CS1410 that had 23 students enrolled. Grades were computed manually by the instructor and independently by AGAF. Grading was compared on two dimensions: time spent grading and similarity of assigned grades. As expected, AGAF dramatically decreases the time to grade, generally by a couple orders of magnitude. The grades assigned by AGAF compare favorably with those of the instructor. Some discrepancies identified AGAF flaws that can be corrected. Other discrepancies identified manual grading errors and inconsistencies. In a few cases, discrepancies are due to AGAF tool limitations thus requiring manual grading of these specific cases. But, even in these situations, the amount of manual grading is reduced significantly, 69-86% in our evaluations. Thus, overall, we have a high degree of confidence in the performance of the AGAF tool suite and anticipate its continued use and development.

6.2 Future Work

One area for the work to expand is in supporting even more of the CS1410 assignments. Word and PowerPoint assignments can leverage C#.NET libraries that are similar to the package that was used for the AGAF Excel grading tool. LinkedIn provides an application programming interface (API) that can be used to enhance the grading of that assignment. In particular, use of the LinkedIn API coupled with assignment modifications could alleviate the missing connection invitation email issue. As other web-based vendors, such as SurveyMonkey, continue to upgrade their services by providing API accessibility options, some of the page scraping AGAF tools can be

upgraded. The Google Drive assignment has students upload a screen capture image to ensure the conversion of uploaded documents. This would likely leverage the Codeacademy image analysis. There is some recent research to reduce the brittleness of page scraping applications [9]. The AGAF web scrapers might be made more stable by incorporating some of these efforts.

Another area for improvement is the user interface of the AGAF tools. Currently, each tool is a standalone application and some have input parameters that are hardcoded into the tools. These could be combined into a single AGAF front-end application that allows the selection of a particular tool and the necessary input parameters and then spawns off a process to run the selected tool. Related to this idea is to have a web-based interface instead of a standalone, desktop application.

The AGAF tools compute a final grade and provide feedback about deductions that were applied. Modifications could be made such that the tool is used more for formative assessment. Students could submit a potential solution to such a modified form of AGAF to learn if they successfully performed the work or what areas are not done correctly. This would turn the grading tool into more of a guide for students helping them to get the correct solution and perhaps making them more motivated to keep working on their approaches to the problems.

Some of the grading tools are very assignment-specific, for example looking for a miles-per-gallon formula. An alternate method that might be investigated further would have the instructor submit a solution file that is compared with student submissions. In this case, instead of looking for a specific formula the tool would be looking for a formula that matches the one in the instructor solution.

A last item of future work could be more extensive checking for plagiarism. The grading tool can save prior submissions and compare against those. There are programming similarity checkers [22] that might yield similar approaches for these types of assignments too.

Bibliography

[1] Aiken, A. Plagiarism Detection. Retrieved December 1, 2015, from Stanford University: https://theory.stanford.edu/~aiken/moss.

[2] At the Forge - Checking Your HTML, 2009. Retrieved November 28, 2015, from Linuxjournal: http://www.linuxjournal.com/magazine/forge-checking-your-html.

[3] Open Source QRCode Library, 2007. Retrieved November 28, 2015, from CodeProject: http://www.codeproject.com/Articles/20574/Open-Source-QRCode-Library.

[4] Codecademy - About the company. Retrieved December 1, 2015, from Codecademy: https://www.codecademy.com/learn.

[5] Balakrishnan Dasarathy, Kevin Sullivan, Douglas C. Schmidt, Douglas H. Fisher, and Adam Porter. The Past, Present, and Future of MOOCs and Their Relevance to Software Engineering. *Proceedings of the on Future of Software Engineering – FOSE*, Hyderabad, India, May 2014.

[6] Stersom, E. Rendering a Web Page – Step by Step, 2010. Retrieved November 28, 2015, from Friendly Bit RSS: https://friendlybit.com/css/rendering-a-web-page-step-by-step.

[7] Expresso Regular Expression Tool. Retrieved November 28, 2015, from Expresso: http://www.ultrapico.com/expresso.htm.

[8] Fielding, R, Gettys, J, Mogul, J, Frystyk, H. Hypertext Transfer Protocol -- HTTP/1.1, RFC 2068, DOI 10.17487/RFC2068, 1997. Retrieved December 8, from World Wide Web Consortium: http://www.rfc-editor.org/info/rfc2068.

[9] Patrick Hagge. Algorithms for Web Scraping. Technical University of Denmark, 2011.

[10] Heidi Han and Ting Liu. The Application of Natural Language Processing and Automated Scoring in Second Language Assessment. *Working Papers in TESOL & Applied Linguistics*, 12(2):20, 2012.

[11] Hedley, J. Jsoup Java HTML Parser, 2009. Retrieved December 1, 2015, from Jsoup: http://jsoup.org/.

[12] David Jackson, and Michelle Usher. Grading Student Programs Using ASSYST. *SIGCSE Bull. ACM SIGCSE Bulletin*, 29(1):233–237, 1997.

[13] Drake Jeanette, Drake Jeffrey, and Michele Ewing. Online Education: Exploring Uses and Attitudes toward Web-Based Learning in Public Relations. *International Journal of Instructional Media*, 37(4):343, December 2010.

[14] Eric Kasten. HTML: a Gentle Introduction. *Linux Journal*, 15(1), July 1995.

[15] Kevin Matthews, Thomas Janicki, and, Laurie Patterson. Implementation of an Automated Grading System with an Adaptive Learning Component to Affect Student Feedback and Response Time. *Journal of Information Systems Education,* 23(1):71-83, Spring 2012.

[16] Andy Kurnia, Andrew Lim, and Brenda Cheang. Online Judge. *Computers & Education*, 36(4):299-315, May 2001.

[17] Microsoft OneNote. Retrieved November 28, 2015, from Microsoft OneNote: https://www.onenote.com.

[18] Andy Nguyen, Christopher Piech, Jonathan Huang, and Leonidas Guibas. Codewebs: Scalable Homework Search for Massive Open Online Programming Courses. *Proceedings of the 23rd International Conference on World Wide Web*, New York, NY, 2014.

[19] Les Perelman. Critique of Mark D. Shermis & Ben Hamner, Contrasting State-of-the-Art Automated Scoring of Essays: Analysis. *The Journal of Writing Assessment*, 6(1), August 2013.

[20] Richard Pregent. *Charting Your Course: How to Prepare to Teach More Effectively*. Atwood Publishing, 2000.

[21] Sameer, A. Web Scraping Tutorial. Retrieved November 28, 2015, from Codediesel: http://www.codediesel.com/php/web-scraping-in-php-tutorial.

[22] Benno Stein, Sven Meyer, Zu Eissen, and Martin Potthast. Strategies for Retrieving Plagiarized Documents. *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* Amsterdam, Netherlands, 2007.

[23] Make Better Decisions with the World's #1 Survey Platform. Retrieved November 28, 2015, from SurveyMonkey: https://www.surveymonkey.com/?ut_source=header.

[24] J Vernon. Attitudes and Opinions of Faculty Tutors about Problem-based Learning. *Academic Medicine*, 70(3):216-23, March 1995.

[25] Andrew Walker. *Essential Readings in Problem-based Learning*. Purdue University Press 2015.

[26] Kevin Warne. Experiences in Implementing Automated Evaluation and Grading in a MOOC Using a Behavior-driven Testing Framework.

[27] Web-CAT. Retrieved December 1, 2015, from Web-CAT: http://web-cat.cs.vt.edu/Web-CAT/WebObjects/Web-CAT.woa.

[28] Welz, M. Free and Easy OCR for C# Using OneNote, 2014. Retrieved December 1, 2015, from Journey of Code: http://www.journeyofcode.com/free-easy-ocr-c-using-onenote.

[29] Williamson, D, Bennett, R, and Lazer, S. Automated Scoring for the Assessment of Common Core Standards, 2010. Retrieved December 1, 2015, from Educational Testing Service: https://www.ets.org/s/commonassessments/pdf/AutomatedScoringAssessCommonCoreStandards.pdf.

[30] Mengmeng Yang, Kai Chen, Zhong Chen Miao, and Xiaokang Yang. Cost-Effective User Monitoring for Popularity Prediction of Online User-Generated Content. *IEEE International Conference on Data Mining Workshop*, Shenzhen, China, 2014.

[31] John Zlatko, and Steven Green. Automated Assessment of Advanced Office Skills. *DEANZ Conference*, Wellington, New Zealand, 2010.

Appendix

All of the programs developed for this thesis are included on the enclosed DVD. Following is a list of all of the files on the DVD:

1. QR Code App: QR grader application developed for this thesis
2. SurveyMonkey App: Grade SurveyMonkey assignments
3. Linkedin App: grade Linkedin assignment
4. Excel grader: Checks formula, and spreadsheet's name and author of the excel
5. OCR grader: Read the texts of the images and find the exact match
6. Blog grader: grade blog pages

Vita

Bahareh Akhtar was born in 1989, in Hamedan, Iran to Nahid and Hossein Akhtar. She graduated from Buali Sina University with a Bachelor of Software Engineering in Hamedan in June 2012. She entered Appalachian State University in January 2013 to pursue graduate studies in Computer Science. During this time she worked as a graduate teaching assistant (GTA). GTA duties included grading student work, assisting with assignment development, and communication with students. A notable pedagogical suggestion was to use student posts to online help forums as extra-credit opportunities. This dramatically improved the volume and quality of student engagement using the forums. In December 2015, Ms. Akhtar received the Master of Science degree in Computer Science from Appalachian State University.